Univerzitet u Beogradu
Matematički fakultet

Marko Šošić

ANALIZA TCP ALGORITAMA ZA KONTROLU ZAGUŠENJA

Master rad

Beograd
2013.

Mentor: Doc. dr Miroslav Marić
Matematički fakultet u Beogradu

Članovi komisije: Prof. dr Aleksandar Jovanović
Matematički fakultet u Beogradu

Prof. dr Duško Vitas
Matematički fakultet u Beogradu

Datum odbrane: _____

ANALIZA TCP ALGORITAMA ZA KONTROLU ZAGUŠENJA

APSTRAKT

TCP je protokol na koji se većina aplikacija oslanja kada želi da prenese podatke preko mreže. On je na Internetu dominantan, gotovo bez konkurencije, zahvaljujući jednostavnosti korišćenja i jakih garancija koje pruža. Međutim, puno aplikacija zahteva brz prenos preko veza sa velikim propusnim opsegom i velikim vremenom obilaska i tu standardni TCP nailazi na poteškoće. On ima veliki problem da postigne optimalne performanse i iskoristi čak i deo kapaciteta. Vremenom, puno rešenja je predloženo. Ovaj rad opisuje standardne TCP algoritme, duge veze sa velikim kašnjenjem velikog kapaciteta i probleme u efikasnosti koji na njima postoje. Dat je pregled pet modernih algoritama za rešavanje istih, kao i rezultati uporednih testova.

SADRŽAJ

1 UVOD

TCP (Transmission Control Protocol) [1] je protokol transportnog sloja TCP/IP modela. Stvoren je 1974. godine kao deo monolitične celine nazvane Transmission Control Program. Kasnije je podeljen na modularnu arhitekturu, TCP/IP model, čiji su glavni delovi TCP i IP (Internet Protocol), a koja se koristi i danas.

Kada aplikacija želi da pošalje podatke preko mreže, dovoljno je (i neophodno) da prosledi zahtev i željene podatke TCP-u. To drastično pojednostavljuje stvari, jer aplikacija ne mora da zna ništa više o mreži, niti da vodi računa o dospeću podataka. Dovoljno je da „vidi“ TCP i njegov interfejs. Šta više, garantovano je da će ti podaci stići aplikaciji na odredištu identični poslatim. TCP garantuje pouzdan prenos, detekciju grešaka, kontrolu toka i zagušenja. Zbog svoje jednostavnosti i jakih garancija, TCP je dominantan transportni protokol. Većina aplikacija, kao što su WWW, email, FTP, SSH, peer-to-peer i druge koriste TCP [2], [3], [4], [5]. Koristi ga gotovo svaki uređaj koji koristi Internet, telefoni, tableti, računari, pa čak i poneki frižider.

Slanje velikih količina podataka preko mreže je nešto što danas zahteva puno aplikacija. Međutim, postizanje efikasnog prenosa preko veze velikog kapaciteta sa velikim vremenom obilaska (vreme neophodno da informacija stigne do odredišta i nazad; eng. Round Trip Time – RTT) postaje nemoguće sa standardnim TCP-om. Jedan od njegovih sastavnih delova, standardni algoritam za kontrolu zagušenja, previše je konzervativan [6], [7] i predstavlja veliki limitirajući faktor čak i kada zagušenja na mreži nema. Drugi problem je nefer alokacija kapaciteta prema vezama koje imaju veliko vreme obilaska [8], [9], [10], [11], za šta je takođe odgovoran isti algoritam.

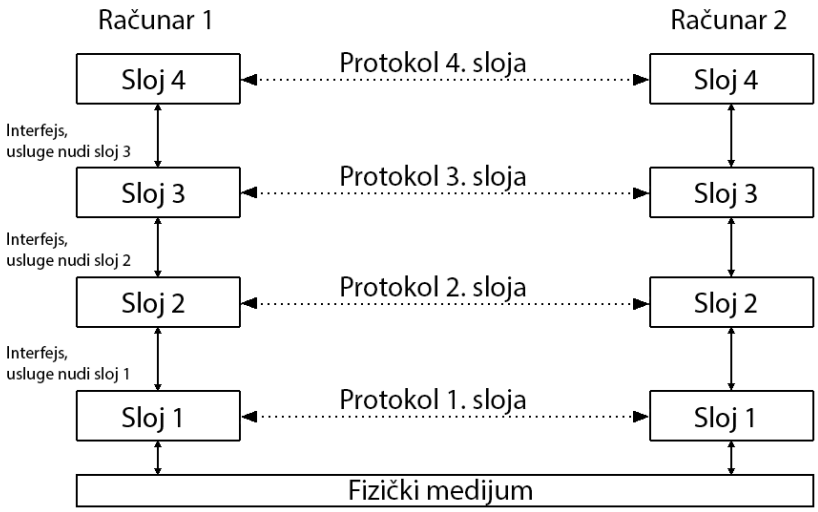
Tekst koji sledi fokusiran je na glavne probleme u efikasnosti na vezama velikog kapaciteta i velikog kašnjenja i pokušaje da se oni reše. Rad je podeljen u celine. Poglavlje 2, Pogled na TCP protokol, opisuje protokol i njegovo mesto u arhitekturi mreže. Uvode se i definišu osnovni pojmovi koji se koriste u narednim delovima rada, kao što su princip krajnjih tačaka, Maks. – Min. fer alokacija, prozor zagušenja, samo zagušenje, i drugi. Poglavlje 3 opisuje klasične algoritme za rešavanje zagušenja, Tahoe i Reno. Dati su osnovni principi prvih algoritama koji se, uz manje ili veće izmene, i danas koriste. Poglavlje 4 prikazuje probleme koji se javljaju na vezama sa velikim kapacitetom i kašnjenjem sa klasičnim algoritmima, pre svega veliki proizvod kapaciteta i vremena obilaska. Peto poglavlje opisuje pet modernih algoritama za rešavanje istih, BIC, CUBIC, Hybla, Vegas i Illinois, a šesto daje eksperimentalnu analizu kroz rezultate uporednih testova. Poslednje, sedmo poglavlje, je zaključak.

2 POGLED NA TCP PROTOKOL

2.1 TCP U ARHITEKTURI MREŽE

Kako bi se projektovanje mreže pojednostavilo, arhitektura je organizovana u zasebne celine koje se obično nazivaju slojevi. Svaki sloj nudi usluge višem sloju, dok je realizacija tih usluga od višeg sloja skrivena. Skup tih usluga zove se interfejs. S druge strane, svaki sloj koristi usluge koje mu pruža sloj ispod. Time se omogućava lakše održavanje, jer su funkcije svakog sloja jasno određene, i svaki se može nezavisno unapređivati sve dok interfejs ostaje isti. Ovaj koncept nije jedinstven za računarske mreže ili za računare i sreće se u mnogim sferama. Npr. vojska ima slojevito upravljanje, ili dobro uređena firma, itd.

Kada se kaže protokol, misli se na skup pravila, tj. način komunikacije između dva sloja na istim nivoima na računarima koji komuniciraju. Npr. TCP preuzima podatke od aplikacije, pakuje ih i prosleđuje dalje. Na drugoj strani, na odredišnom računaru, TCP sloj ume te podatke da prepozna, raspakuje i prosledi naviše onakve kakvi su poslati, zahvaljujući uspostavljenom dogovoru koji se poštuje, tj. protokolu. Princip protokola je veoma važan, jer, u slučaju mreža, omogućava međusobnu komunikaciju između najrazličitijih uređaja dok god poštuju iste protokole.



Slika 1: Odnos slojeva i protokola – svaki sloj nudi usluge sloju iznad kroz interfejs

2.1.1 TCP/IP model

Skup slojeva i protokola čini arhitekturu mreže. Danas najzastupljenija arhitektura nosi ime TCP/IP referentni model [12], kao kombinacija imena njegova dva osnovna protokola. Sam model se sastoji od četiri sloja: aplikacijskog, transportnog, internet sloja i sloja veze.

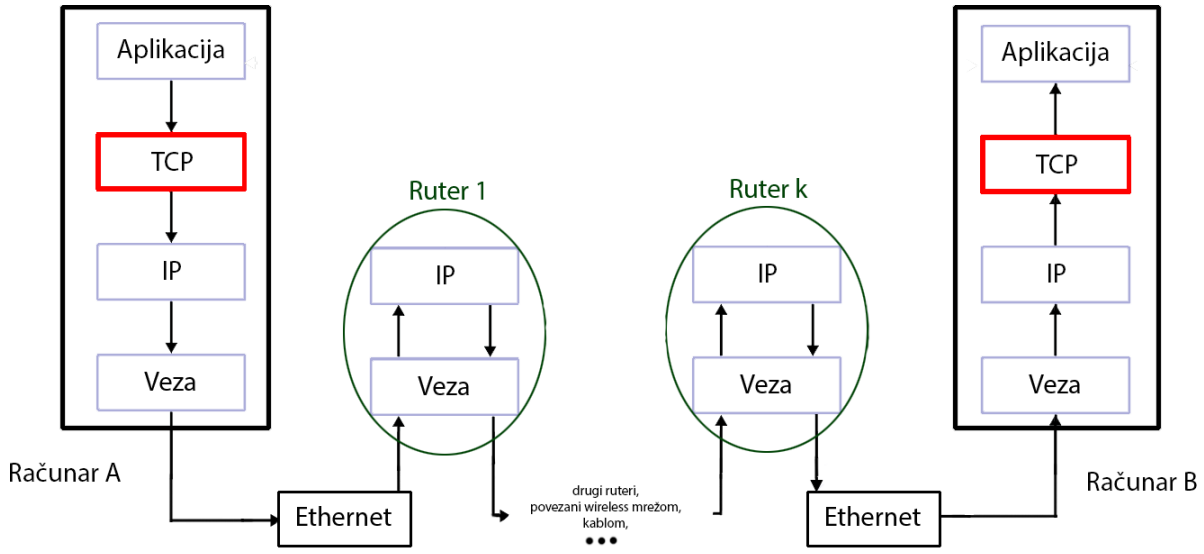
Sloj aplikacije definiše konkretna aplikacija koja koristi mrežu. Dakle, ako aplikacija ima zadatak da šalje poruke elektronskom poštom, ona može implementirati skup pravila već

definisanih protokolom SMTP (eng. Simple Mail Transfer Protocol) ili nekim drugim, proizvoljnim. Jedini uslov je da aplikacije koje komuniciraju koriste identičan protokol. Neki poznatiji aplikacijski protokoli su HTTP (eng. Hypertext Transfer Protocol), POP (eng. Post Office Protocol), DHCP (eng. Dynamic Host Configuration Protocol), SSH (eng. Secure Shell).

Transportni sloj je ispod aplikacijskog. Glavni predstavnik je TCP, na kome će se fokusirati sledeća poglavlja. On je zadužen da tok podataka, primljen od aplikacije, podeli u pakete, svaki odgovarajuće obeleži, i obezbedi da aplikacija kojoj se šalje dobije isti tok podataka i istim redosledom kojim su poslati. Vredi pomenuti još jedan protokol koji je u upotrebi, UDP (eng. User Datagram Protocol), koji ne pruža nikakve garancije o kvalitetu podataka nakon što se prenesu preko mreže.

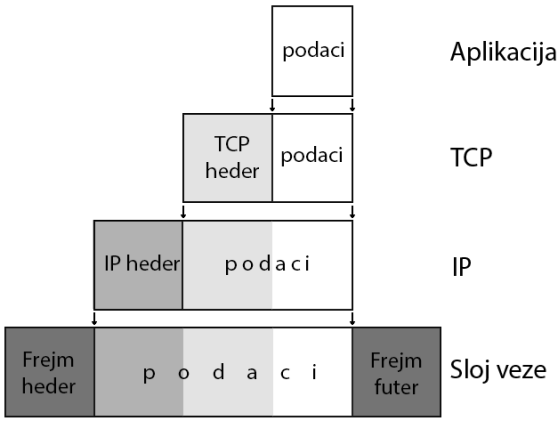
Internet sloj, konkretno IP (eng. Internet Protocol), ima zadatak da pakete koje računari prosleđuju mreži uputi na odredište. Svaki čvor ima jedinstvenu adresu na mreži, takozvanu IP adresu. Paketi „skakuću“ (eng. Hop-by-hop Delivery) kroz čvorove mreže dok ne stignu na odredište. Oni mogu stići drugačijim redosledom, ili ne stići uopšte; IP ne daje nikakve garancije, već samo pruža najveći mogući napor u dostavljanju paketa (eng. Best-effort Delivery). Ovde je dato vrlo kratko objašnjenje IP-a, s obzirom na to da on nije glavna tema rada.

Neki slojevi veze su Ethernet, DSL (eng. Digital Subscriber Line), PPP (eng. Point-to-point Protocol) i drugi.



Slika 2: TCP/IP model i skica veze između dva računara sa ruterima između

Na izvorišnom računaru, svaki sloj podatke „pakuje“ tako što doda potrebne informacije u zaglavlje (heder) i to prosledi sloju niže. Podatke na destinaciji dobija prvo najniži sloj, koji ih otpakuje, pročita svoj heder, ukloni ga i ostalo prosledi višem sloju. Na kraju podaci stižu do aplikacije „otpakovani“. Ovo se zove umotavanje podataka ili enkapsulacija.



Slika 3: Enkapsulacija podataka – svaki sloj upakuje podatke i prosledi sloju niže

2.1.2 Princip krajnjih tačaka

Aplikacijski i transportni sloj se, gotovo isključivo, nalaze samo na krajnjim čvorovima (računarima) koji komuniciraju, tj. ne postoje na čvorovima (ruterima) između, koji ih povezuju. Ovakva uređenost je vrlo bitna i princip se zove princip krajnjih tačaka (eng. End-to-end Principle) [13]. On omogućuje da se od same mreže zahteva vrlo malo (takozvani nizak ulazni prag). Time se dozvoljava da mreža bude veoma heterogena, a istovremeno fleksibilna kada je funkcionalnost u pitanju, jer je najveći deo funkcionalnosti, viši slojevi, na krajevima. Oni su najodgovorniji za podatke koje prenose, dok mrežu posmatramo kao apstrakciju najboljeg napora, bez garancije, da se individualni paketi dostave. Od mreže se, dakle, zahteva da podrži internet sloj i sloj veze.

2.2 POUZDAN PRENOS

Način na koji je mreža projektovana dozvoljava pojavu greške ili gubitak poslatih podataka. Fizički medijum, čak i najmoderniji, ne može u potpunosti eliminisati smetnje, pa se samim tim, ma koliko verovatnoća greške bila mala, ne može garantovati ispravnost dospelih podataka na tom nivou. Šta više, može doći do potpunog prekida veze (npr. prekinemo kabl), što nije retka pojava. Drugo, može se desiti da se podatak izgubi iz razloga koji nije fizičke prirode, kao što je preopterećenje nekog rutera na putu do odredišta, ili preopterećenje primaoca. IP ne obezbeđuje nikakvu pouzdanost. Dakle, mora se uzeti u obzir da poslati podatak nije isto što i primljen, i napraviti neku vrstu potvrde koji šalje primalac. Upravo to je osnova TCP-ovog pouzdanog prenosa, što će biti opisano u daljem tekstu.

TCP na početku otvara konekcije između računara koji komuniciraju, tj. uspostavlja direktnu vezu između njih mehanizmom trostepenog usaglašavanja (eng. 3-way handshake). Jedna strana je pasivna (server), i ona čeka zahtev za konekcijom. Druga je aktivna (klijent), i ona šalje zahtev. Po uspostavljenoj vezi, svaka strana je inicijalizovala stanje, koje održava tokom konekcije. Stanje čini skup promenljivih koje čuvaju podatke važne za funkcionisanje konekcije. Npr. IP adresa i port su podaci koji se čuvaju, ili redni broj paketa i sl. Ovde je zgodno reći da se

pomenuta stanja održavaju samo na krajnjim čvorovima. Ruteri u mreži ih ne održavaju, ne prate pojedinačne tokove, već pakete vide kao celine za sebe.

TCP podatke deli u pakete veličine do 64KB. Oni su uglavnom dosta manji od 64KB, obično veličine 1460B, zbog ograničenja veličine Ethernet frejma. Svaki se obeležava rednim brojem (informacija koja se čuva u hederu), i prosleđuje dalje IP protokolu, pa protokolu veze, i mrežom putuje do destinacije. Sve što primalac (TCP na odredištu) treba da uradi da bi utvrdio da li neki od paketa nedostaje ili je stigao van redosleda ili u duplikatu jeste da pogleda redne brojeve. Duplikati i paketi van redosleda se trivijalno rešavaju.

Primalac za dospele pakete šalje potvrdu (koju ćemo nadalje zvati i ACK – eng. Acknowledgement), kako bi pošiljalac znao da su oni uspešno dostigli odredište. Za to se koristi mehanizam kumulativne potvrde (eng. Cumulative Acknowledgement). Umesto da se za svaki paket šalje potvrda, šalje se samo redni broj (ACK) sledećeg paketa koji se očekuje, tako da su svi pre njega uspešno primljeni. Npr. ako pošiljalac pošalje pakete sa rednim brojevima: 1, 2, 3, 4, 5 i 6., a stignu paketi: 1, 2, 3 i 6, primalac će poslati ACK=4, jer su svi pre 4. uredno stigli. Primalac će slati potvrdu ACK=4, sve dok 4. ne stigne, bez obzira da li su paketi sa većim rednim brojevima dospeli ili ne. Time se pošiljalac signalizira da 4. paket fali i on će biti ponovo poslat.

Ofset	Oktet	0								1								2								3							
Oktet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Port izvora																Port destinacije															
4	32	Redni broj paketa																															
8	64	ACK broj (broj potvrde)																															
12	96	Ofset				Rezerv. 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Veličina kliznog prozora															
16	128	Kontrolna suma																Pokazivač za hitni paket															
20 ...	160 ...	Opcije ...																															

Slika 4: TCP heder – podaci i opcije koji se prenose svakim paketom

Može se desiti da se potvrda izgubi u putu. Da bi se sprečilo večno čekanje na potvrdu, pošiljalac za svaki od paketa pri slanju pokreće tajmer. Kada određeno vreme istekne, a potvrde nema, paket se ponovo šalje.

Ovakav mehanizam garantuje isporuku svih paketa.

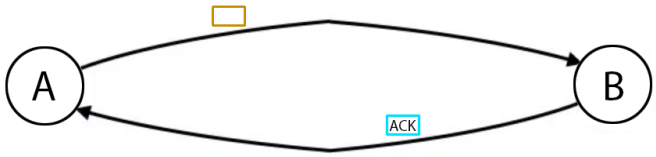
2.3 Detekcija grešaka

Paket koji stigne može biti oštećen, tj. može doći do izmene podataka usled šuma pri prenosu. Da bi se greške detektovale koristi se kontrolna suma (eng. checksum) [1]. Ceo TCP

paket se deli u 16-bitne reči. Te reči se saberu u komplementu jedinice i komplement jedinice zbira je kontrolna suma. U trenutku sabiranja, polje kontrolne sume je popunjeno nulama. Kada paket stigne, primalac sabira sve 16-bitne reči, uključujući i kontrolnu sumu, i ako dobije 16 nula kao rezultat to znači da je paket ispravan. Ako je paket neispravan, odbacuje se i postupak je isti kao da uopšte nije ni stigao.

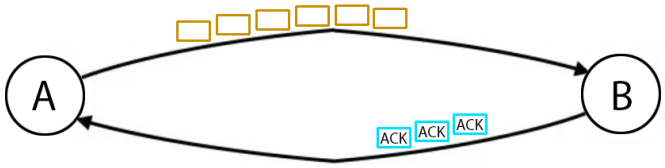
2.4 KONTROLA TOKA

Za prenos paketa od A do B možemo koristiti sledeći jednostavan metod. A pošalje paket i čeka da dobije potvrdu (ACK). Kada potvrda stigne, on šalje novi paket. Ako potvrda ne stigne, A čeka određeno vreme i nakon toga ponovo šalje isti paket. Ovaj metod se zove „stani i čekaj“ (eng. Stop and Wait). On služi samo kao najjednostavniji primer i u praksi se ne koristi, jer bi bio vrlo neefikasan, evo zašto. Neka je vreme obilaska između A i B 50ms, a propusni opseg veze 10Mb/s. Pretpostavimo da se nijedan paket ne izgubi (najbolji mogući slučaj), i da šaljemo npr. Ethernet pakete, koji su veličine 1,5KB, tj. 12 Kb. Svake sekunde može se poslati najviše 1000ms/50ms = 20 paketa. Dakle, brzina prenosa bi bila 20*12Kb = 240Kb = 0,24 Mb. To je ~2,4% ukupne propusne moći veze.



Slika 5: „Stani i čekaj“ – samo jedan paket je „u letu“, tj. dozvoljeno je poslati samo jedan paket pre nego što stigne potvrda – jako neefikasno

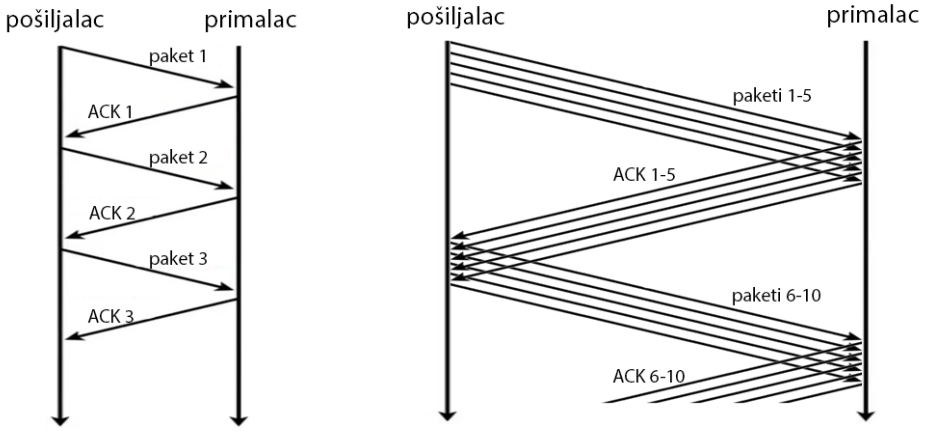
Prethodni pristup je očigledno neupotrebljiv, ali njegova generalizacija, tehnika kliznog prozora (eng. Sliding Window), vrlo je uspešno korišćena. Umesto da se šalje jedan paket i čeka potvrda, šalje se određeni broj paketa (prozor) istovremeno. Ovo omogućava efikasan prenos.



Slika 6: Klizni prozor – više paketa se šalje pre nego što stignu potvrde – efikasan način

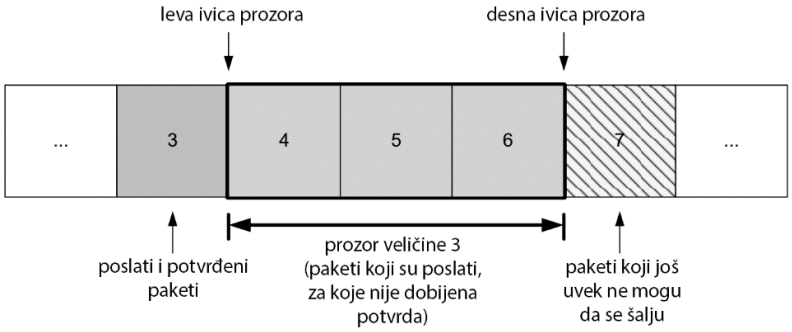
TCP održava tri promenljive: veličinu kliznog prozora (eng. Send Window Size – SWS), broj poslednje potvrde (eng. Last Acknowledgment Received – LAR) i broj poslednjeg poslatog paketa (eng. Last Segment Sent – LSS). U svakom trenutku važi LSS – LAR ≤ SWS, tj. razlika broja

poslednjeg poslatog i poslednjeg potvrđenog paketa je najviše veličina kliznog prozora. Time se ograničava broj paketa koji su „u letu“ (eng. data in flight) u svakom trenutku. Na svaki novi ACK LAR se osvežava na redni broj te potvrde.



Slika 7: Uporedno poređenje efikasnosti „stani i čekaj“ metode i kliznog prozora – iako je vreme obilaska (razmak između, na slici) veće, klizni prozor (desno) uspeva da pošalje više podataka nego „stani i čekaj“ (levo)

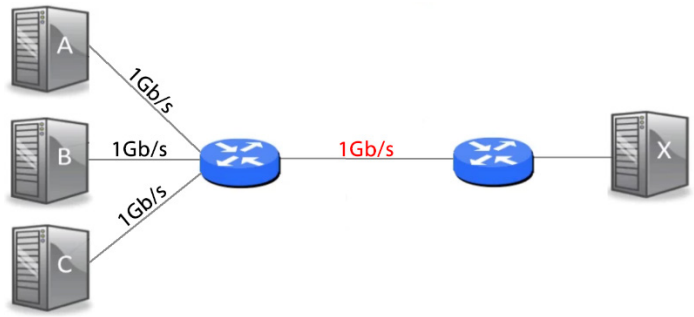
Koliki treba da bude klizni prozor? Šta se dešava ako primalac nije u stanju da obradi podatke koji mu se šalju određenom brzinom? Npr. računar koji prima podatke je dosta sporiji, opterećeniji, ... U takvoj situaciji, paketi se jednostavno gube i bivaju poslati ponovo. Svaki podatak koji se pošalje, a ne stigne, usled preopterećenosti primaoca, jeste nepotrebno popunjavanje kapaciteta veze, i efikasnije bi bilo ne poslati ga. Primalac mora da signalizira kojom brzinom je u stanju da prima, kako se nikada ne bi slalo brže od toga. TCP u zaglavlju sadrži polje za veličinu prozora, koju određuje (često se koristi izraz reklamira, eng. advertise) primalac i koju pošiljalac mora poštovati. Dakle, veličina kliznog prozora dobija vrednost veličine prozora koju reklamira primalac (eng. Advertised Window - awnd). Ovo se zove kontrola toka, jer se tok podataka prilagođava primaocu tako da nema nepotrebnih gubitaka i ponovnih slanja.



Slika 8: Klizni prozor (ime je proizašlo iz činjenice da se u određenom trenutku vidi samo deo, prozor, paketa) – levo su poslati i potvrđeni paketi, desno su paketi koji se ne mogu poslati dok se ne dobije sledeća potvrda, a u sredini oni koji se trenutno šalju

2.5 ZAGUŠENJE

Za sada je pokazano kako se može rešiti problem sporog primaoca. Ali šta se dešava kada su obe krajnje tačke brze, a mreža između njih nije u stanju da prenese podatke brzinom koju diktiraju? Npr. neka su računari povezani kao na slici 9. Svaki računar može postići brzinu od 1Gb/s, i veza do odredišnog računara je takođe 1Gb/s. Ako bi svaki računar (levo) slao podatke svojom maksimalnom brzinom (ukupno 3Gb/s), to bi prekoračilo maksimum koji veza može podneti. U tom slučaju dve trećine paketa bi se gubile.

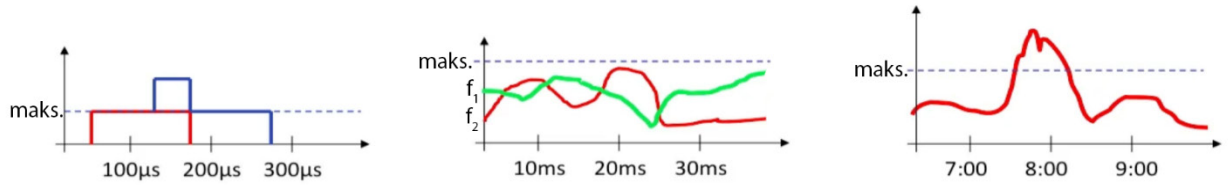


Slika 9: Primer potencijalnog zagušenja mreže – npr. kancelarija sa spoljnom vezom od 1Gb/s i tri računara u njoj, svaki sa vezom od 1Gb/s do prvog rutera

Problem je sličan problemu sporog primaoca, i kao i tamo, ovde treba reagovati i odrediti maksimalnu brzinu koju veza može podržati u datom trenutku i nju koristiti kao gornju granicu. Kada se kaže brzina mreže ili veze, obično se misli na brzinu rutera (fizički medijum, npr. kabl, uglavnom nije limitirajući faktor). Ruter ima ograničenu brzinu kojom prosleđuje pakete. Ako mu se dostavi više paketa nego što je u stanju da prosledi, oni se ne odbacuju odmah, već se smeštaju u bafer za kasniju obradu. Međutim, ma koje veličine da je bafer, ako ruter neprekidno prima više podataka nego što je u stanju da obradi, bafer će se napuniti i naredni paketi će biti odbačeni. Dakle, zagušenje je stanje kada se paketi gube usled nesposobnosti mreže da ispuni zahteve krajnjih tačaka određenom brzinom. Zagušenjem se može smatrati i povećano kašnjenje paketa usled punih bafera u ruterima. Ako bi se problem zagušenja ignorisao, nakon nekog vremena većina paketa bi bivala odbačena i ponovo slata, i većinu protoka bi činili paketi koji bi se „negde usput“ gubili usled preopterećenja. Mreža bi tada uspevala da uspešno prenese tek mali deo svog potencijala. Takvo stanje se zove kolaps usled zagušenja (eng. congestion collapse) [14].

Bitna razlika u odnosu na problem sporog primaoca je što ne postoji jasan indikator zagušenja, niti se može dobiti jasna poruka o brzini kojom smemo da šaljemo. Mreža, u generalnom slučaju, ne pruža nikakvu povratnu informaciju i sistem se u potpunosti oslanja na princip krajnjih tačaka. Dakle, ako je potrebno da se zna da li je zagušenje u toku, to mora da „javi“ primalac. Na prvi pogled deluje iznenađujuće da primalac, kao krajnja tačka, obaveštava o zagušenju. On to i ne radi direktno, jer o stanju mreže „zna“ isto koliko i računar koji šalje, već se kao indikator zagušenja obično uzima nedospeli paket. Ako se paket izgubi i ne stigne na odredište, smatra se da je neki bafer usput prepunjen, tj. zagušenje je u toku, i TCP mora da

reaguje adekvatno. Ovo je klasičan način detektovanja zagušenja (takozvani reaktivni način, jer reaguje na gubitak), i TCP algoritmi koriste ovu tehniku još od osamdesetih godina. Postoji i drugačiji pristup gde se meri kašnjenje svakog paketa, i povećano kašnjenje znači dolazeće zagušenje (proaktivni način, jer reaguje pre nego što se zagušenje desi).



Slika 10: Primeri zagušenja: levo – trenutno zagušenja kada dva paketa stižu istovremeno u ruter, jedan će morati da sačeka u baferu; sredina – dva toka se takmiče a njihov ukupan protok je veći od maksimalnog koji ruter može da podrži; desno – previše korisnika koristi vezu u isto vreme

Suprotan problem, koji takođe spada u problem zagušenja u širem smislu, jeste detektovati povećanje propusnog opsega, tj. kada možemo da povećamo brzinu slanja jer je veza dobila veći kapacitet. Razlog zbog kog se i ovo naziva problemom zagušenja, jeste da kada se kaže da TCP reaguje i prilagođava se zagušenju, misli se na prilagođavanje u oba smera, usporavanje na znak zagušenja i ubrzavanje kada zagušenja nema.

TCP održava promenljivu koja se zove prozor zagušenja (eng. Congestion Window – *cwnd*) [15]. Prozor zagušenja je analogon prozoru koji reklamira primalac pri kontroli toka, samo u kontroli zagušenja, umesto brzine primaoca, prozor zagušenja je procena kapaciteta veze, tj. on određuje gornju granicu za broj paketa koji se mogu poslati a da se veza ne preoptereti. TCP uzima u obzir i ovaj parametar, pa je klizni prozor (*swnd*), broj paketa koji TCP šalje, zapravo, jednak minimumu prozora zagušenja (*cwnd*) i reklamiranom prozoru (*awnd*), formula (1). Na taj način, TCP šalje brzinom koji može da podrži primalac ili mreža, šta god da je manje. U praksi je uglavnom prozor zagušenja manji, pa on diktira brzinu.

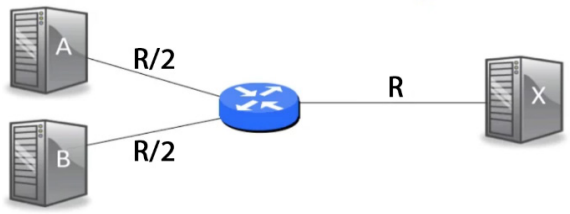
$$swnd = min(awnd, cwnd) \tag{1}$$

Stanje na mreži se vremenom menja, pa se i prozor zagušenja dinamički menja u zavisnosti od toga. Prozor ne sme biti premali, jer se tada veza ne koristi efikasno (videti „stani i čekaj“ primer u kontroli toka, tu je prozor veličine jedan), ali ne može biti ni prevelik, jer tada dolazi do zagušenja i gubitka paketa. Zbog nedostatka direktnog signala, vrednost pomenutog prozora se određuje empirijski. Nije jasno definisano kako se prozor zagušenja računa, pa su se tokom godina razvili mnogi algoritmi [16], koji uzimaju u obzir različite parametre. Dakle, kontrola zagušenja je algoritam koji određuje veličinu prozora zagušenja na osnovu datih parametara. To je obično zasebni modul, koji prima informacije, kao što su redni brojevi paketa, redni brojevi

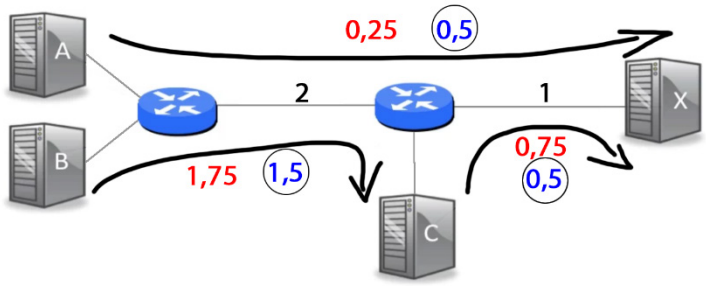
potvrda, itd., računa najbolju vrednost za prozor zagušenja u tom trenutku, i vraća je glavnom TCP modulu.

2.5.1 Maks. – Min. fer alokacija

Kada se više tokova takmiči za dati kapacitet veze, poželjno je da se taj kapacitet deli na jednake delove, fer. Maks. – Min. fer alokacija [17] generalno je prihvaćena kao definicija fer alokacije. Po njoj, fer podela je postignuta ako se ne može povećati udeo u kapacitetu jednog toka, a da se time ne smanji udeo drugog čiji je udeo manji ili jednak. Algoritam za kontrolu zagušenja je dužan (poželjno je) da obezbedi ovakvu vrstu fer alokacije među tokovima.



Slika 11: Trivijalna fer podela na dva jednaka dela – ukupni kapacitet je R; takmiče se dva toka, R se deli na polovine svakom



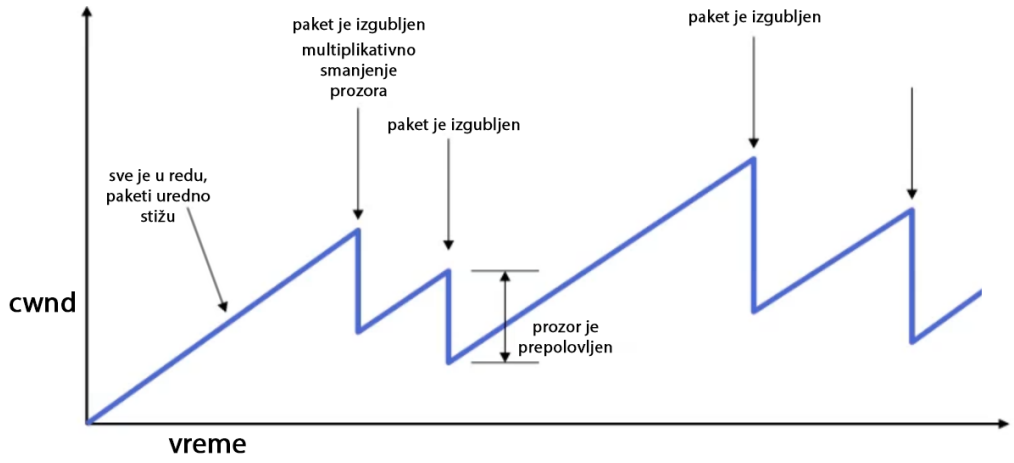
Slika 12: Kompleksiji primer podele kapaciteta: kapacitet svake veze je označen brojem iznad; ako toku A-X dodelimo 0,25, toku B-C 1,75, a toku C-X 0,75, nije izvršena fer dodela, jer se tok A-X može povećati bez smanjivanja manjeg toka; druga podela (zaokruženi brojevi) bi bila dodeliti toku A-X 0,5, toku B-C 1,5 i toku C-X 0,5 i to jeste Maks. – Min. fer alokacija, jer se ne može povećati nijedan tok a da se ne utiće na smanjenje drugog koji je manji ili jednak; primetiti da je ukupan protok kroz mrežu u prvoj podeli veći (2,75) u odnosu na drugu (2,5)

3 KLASIČNI ALGORITMI

3.1 AIMD

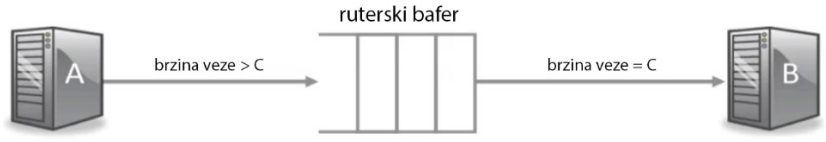
Do sada nije objašnjeno kako se prozor zagušenja može računati. TCP, za to, generalno koristi šemu AIMD (eng. Additive Increase, Multiplicative Decrease – aditivno uvećanje, multiplikativno smanjenje) [15]. Ovo nije potpun algoritam, ali je njegov glavni deo. Sastoji se u sledećem. Ako je paket dospeo, prozor zagušenja se uvećava za konstantu $\alpha/cwnd$, a ako se paket izgubi, prozor se smanjuje multiplikativno za određeni faktor, β, formula (2).

$$cwnd = \begin{cases} cwnd + \frac{\propto}{cwnd}, na\ ACK \\ \frac{cwnd}{\beta}, na\ izgubljeni\ paket \end{cases} \quad (2)$$



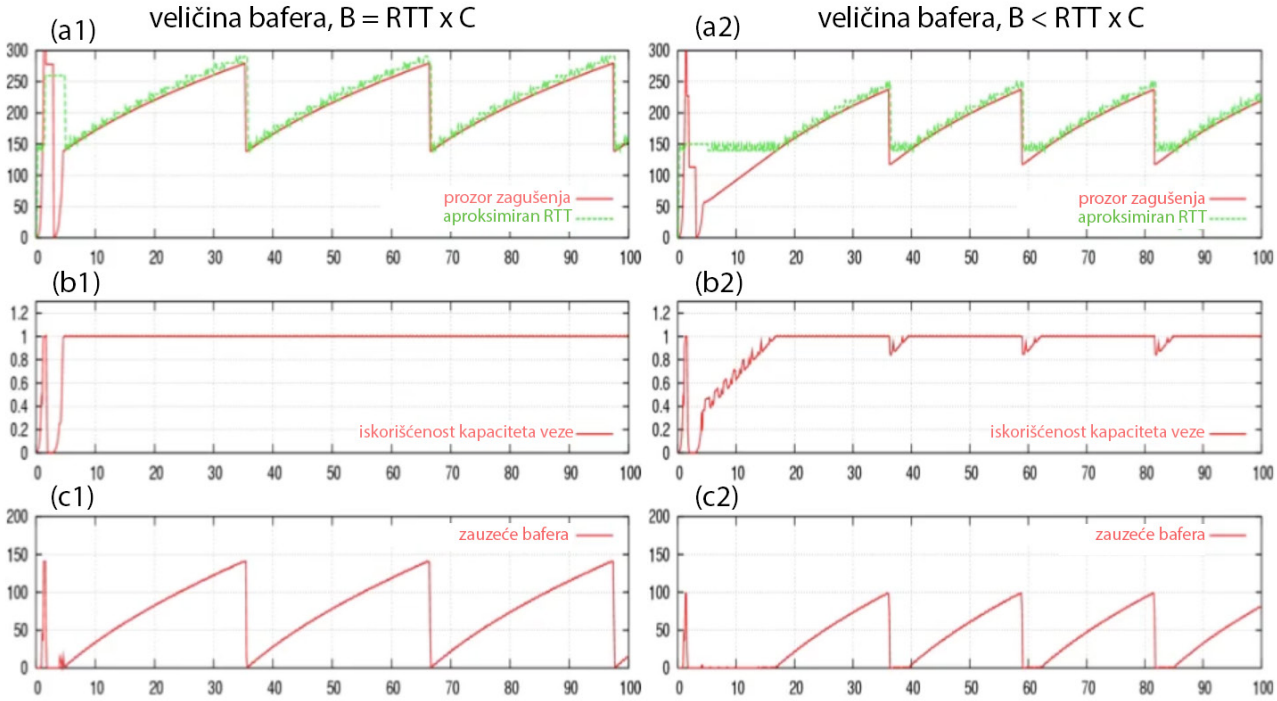
Slika 13: AIMD ponašanje – kada je sve u redu, prozor se na svaki dospeli paket uvećava za konstantu; kada se paket izgubi, prozor se smanjuje multiplikativno (ovde je β=2); oblik ovakvog grafika se često naziva testerasti oblik (eng. Sawtooth Wave)

Neka je data situacija kao na slici 14. Imamo jedan tok podataka kroz jedan ruter (i njegov bafer). Neka se ruteru šalju podaci brže nego što on može da ih prosledi (brzina veze pre rutera je veća od *C*, gde je *C* brzina izlazne veze).



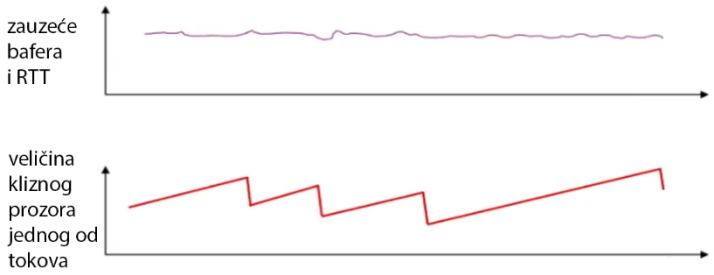
Slika 14: Primer jednog toka kroz jedan ruter i njegov bafer

Na slici 15 dati su grafici ponašanja AIMD šeme u situaciji sa slike 14, gde je α=1, a β=2. Brzina kojom paketi uspevaju da stignu na odredište je određena sa $R = W/\text{RTT}$, gde je W veličina kliznog prozora. Treba primetiti da, pošto je brzina kojom se šalje veća od brzine rutera, kako se bafer rutera puni i u njemu čeka više paketa, vreme obilaska se uvećava. U jednom trenutku, bafer će se prepuniti, i paket će biti izgubljen. Tada se prozor smanjuje, što daje vremena baferu da se isprazni. Klizni prozor se uvećava u direktnoj proporciji sa vremenom obilaska (slika a1). Dakle, W/RTT je konstanta, (slike a1, b1 i c1), što znači da je efektivna brzina uvek konstantna, što se i želi. Ako bi bafer bio manji od RTT x C (slike a2, b2 i c2), to bi značilo da on neće moći da smesti RTT x C količinu paketa, brže će se prepuniti (slika c2), pa W/RTT neće uvek biti konstanta (slika b2). Tada se potencijal veze ne bi koristio do kraja.



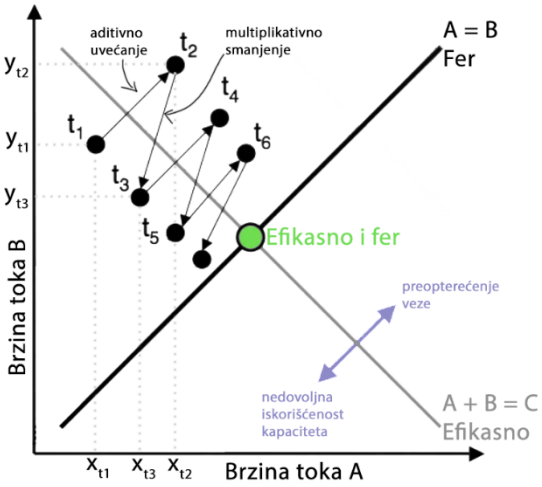
Slika 15: Simulacija ponašanja prozora zagušenja, vremena obilaska, iskorišćenosti kapaciteta veze i zauzeća bafera pri jednom toku kroz jedan ruter; levo – veličina bafera je jednaka RTT x C (brzina rutera); desno – veličina bafera je manja od RTT x C

Neka je sada, za razliku od prethodnog primera, dat ruter i puno tokova koji prolaze kroz njega. U praksi ruteri često imaju i hiljade tokova istovremeno. Prethodni primer sa jednim tokom, dat je za ilustraciju i bolje razumevanje dinamike mreže kroz poređenje. Za razliku od njega, ovde (i na javnoj mreži) jedan tok ne utiče značajno na zauzeće bafera, pa samim tim ni na RTT (slika 16). Dakle, možemo smatrati da je vreme obilaska konstantno. Iz toga sledi da je se brzina kojom dopremamo pakete menja sa veličinom prozora u direktnoj proporciji. Taj zaključak jeste intuitivan. Što je prozor veći, više paketa šaljemo, pa je količina podataka po jedinici vremena veća.



Slika 16: Više tokova kroz ruter; RTT i zauzeće bafera su uglavnom isti, pojedinačni tok ne utiče puno

Suština AIMD principa je da isprobava kapacitet veze polako, uvećavajući prozor i ne stvarajući zagušenje. Na pojavu zagušenja reaguje, i smanjuje brzinu toka u trenutku. AIMD takođe obezbeđuje Maks. – Min. fer alokaciju (kada je RTT tokova isti ili približan), jer multiplikativno smanjenje utiče više na tokove sa većom brzinom, nego na one sa manjom. Slika 17 prikazuje uticaj AIMD šeme na dva toka, A i B, koji se takmiče za kapacitet, gde tok B u startu dosta brže šalje podatke (tačka t_1). Tokovi aditivno uvećavaju svoje prozore (a time i brzinu) do tačke t_2, gde se veza preoptereti i oni izvršavaju multiplikativno smanjenje. Sve iznad sporedne dijagonale je preopterećenje veze, a dijagonala je željena maksimalna iskorišćenost. Nakon toga su u tački t_3 (primetiti da je multiplikativno smanjenje uticalo više na brži tok B), gde je A uvećao brzinu, a B smanjio u odnosu na početnu tačku t_1. Glavna dijagonala predstavlja jednake brzine tokova, i to je željeni rezultat, takođe. Vremenom, tokovi konvergiraju preseku dijagonala, ispunjavajući oba cilja.



Slika 17: Uticaj AIMD šeme na dva toka A i B koji se takmiče za kapacitet (Chiu Jain grafik)

Nedostatak ove šeme je što drastično kažnjava tokove sa velikim vremenom obilaska, što nije direktna namera njenog dizajna, već nesrećna posledica. O ovome će biti više reči u poglavlju o problemima (poglavlje 4).

3.2 TAHOE

Pre nego što je u TCP ugrađen mehanizam kontrole zagušenja, korišćena je samo kontrola toka. Na početku konekcije, TCP bi poslao ceo klizni prozor paketa i za svaki bi pokrenuo tajmer. Ako bi tajmer istekao, a potvrda za paket nije stigla, paket bi se slao ponovo. To je bio osnovni mehanizam slanja. Međutim, krajem osamdesetih godina Internet je bio pred kolapsom. Glavni razlog je bio agresivno ponovno slanje izgubljenih paketa, bez ikakve kontrole, što je samo pogoršavalo stvari i dovodilo mrežu u stanje stalnog zagušenja. Gubitak paketa je bio masivan, a efektivna brzina mala. Prvi algoritam osmišljen da se bori sa ovim problemom je Tahoe [15]. Zgodno je detaljno pogledati tehnike koje on koristi, jer se iste koriste i danas, uz razne modifikacije.

Tahoe je uveo tri poboljšanja, već pomenuti prozor zagušenja, bolju RTT estimaciju, uspomoć koje se postavlja trajanje tajmera za čekanje na potvrdu, i samotempiranje (eng. Self-clocking). Kontrola zagušenja se deli na dva stanja, spori start i izbegavanje zagušenja.

3.2.1 Spori start

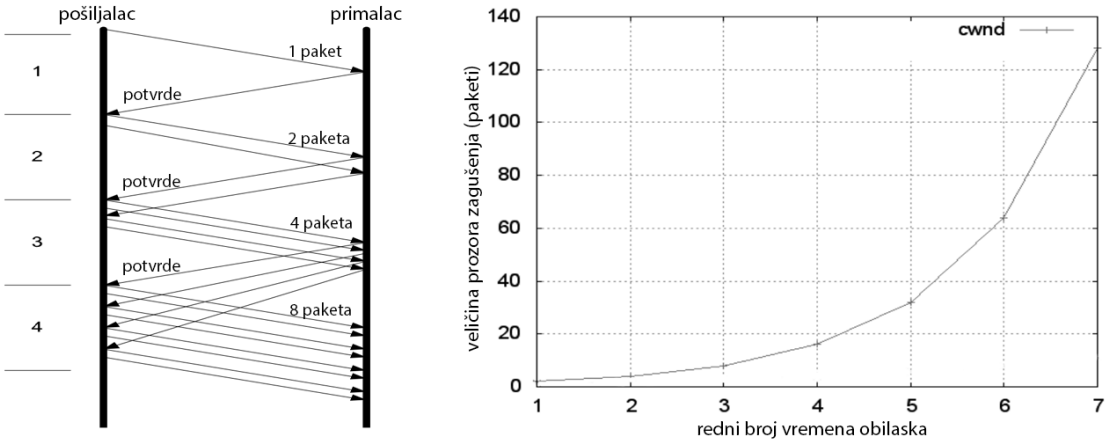
Kada se uspostavi nova TCP konekcija, ona ne može znati koja bi najbolja vrednost za prozor zagušenja (*cwnd*) mogla biti, jer ne zna koliki je kapacitet veze koju koristi. Bez bilo kakvog nagoveštaja za veličinu prozora, ne ostaje ništa drugo nego da se isproba sa slanjem sve brže i brže dok se neki paket ne izgubi, što bi bio indikator zagušenja. Ovo zovemo spori start (eng. Slow Start – SS) [18].

Spori start počinje tako što se na početku pošalje jedan paket i čeka se potvrda. Po potvrdi, šalju se dva paketa i čeka se njihova potvrda, pa četiri, pa osam, itd. Na svaku potvrdu prozor zagušenja se uvećava za jedan, formula (3). Time se dobija eksponencijalan rast prozora na svaki RTT, pošto se za jedno vreme obilaska veličina prozora duplira.

$$cwnd = cwnd + 1 \tag{3}$$

Ovim se efikasno isprobava kapacitet veze. Naziv spori start potiče od poređenja sa ranijim ponašanjem, gde bi se odmah na početku slao ceo klizni prozor, a ovde se počinje sporo, od male veličine prozora, iako ona raste eksponencijalno.

Ma koliki kapacitet da nam je pružen, u jednom trenutku će prozor zagušenja narasti toliko, da će preopteretiti vezu, i paket će biti izgubljen. To je znak da je dostignuta, makar približno, maksimalna brzina.



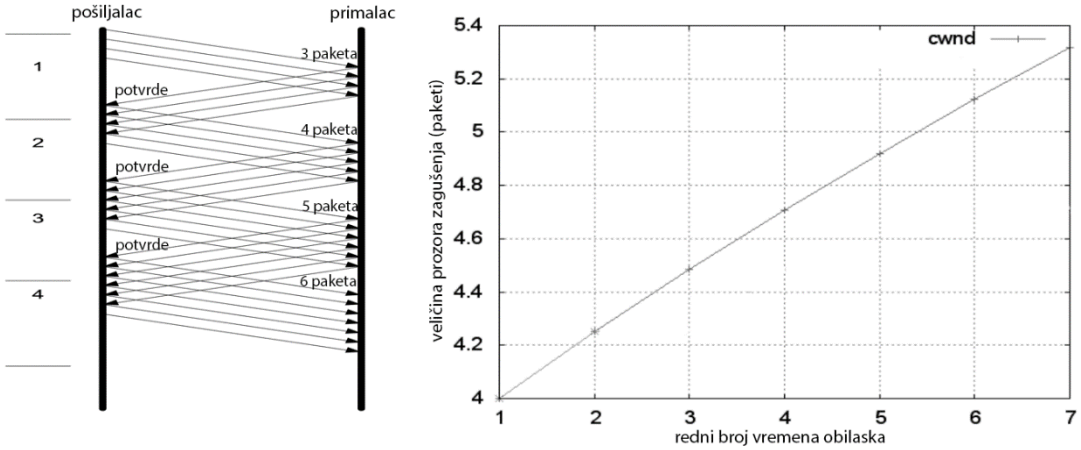
Slika 18: Faza sporog starta; svaka potvrda koja stigne omogućava da se prozor poveća za jedan, pa se on duplira u veličini na svaki RTT

3.2.2 Izbegavanje zagušenja

Kada faza sporog starta grubo utvrdi maksimalnu brzinu, TCP počinje fazu izbegavanja zagušenja (eng. Congestion Avoidance – CA) [18]. Na svaki dobijeni ACK, *cwnd* se uvećava za 1/*cwnd*, formula (4).

$$cwnd = cwnd + \frac{1}{cwnd} \tag{4}$$

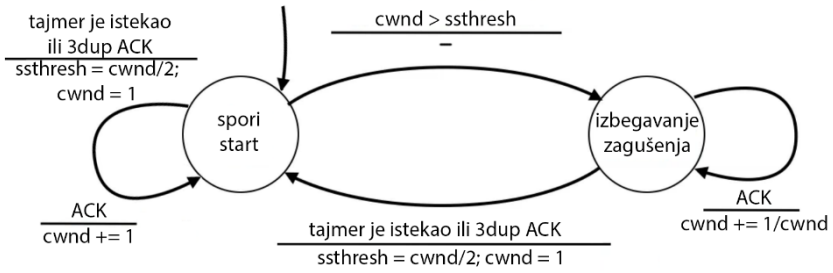
Tako se prozor uvećava za jedan na svaki RTT, što daje linearnu funkciju rasta. Ovo je aditivno uvećanje (videti 4.1 AIMD), gde je α=1. Time se omogućava neagresivan, lagan način uvećanja brzine kako bi se obezbedio veći protok. Ova faza može da bude jako neefikasna (videti poglavlje 4).



Slika 19: Faza izbegavanja zagušenja; svaka potvrda omogućuje da *cwnd* poraste za 1/*cwnd*, pa se on uvećava za jedan na svaki RTT

3.2.3 Prebacivanje između sporog starta i izbegavanja zagušenja

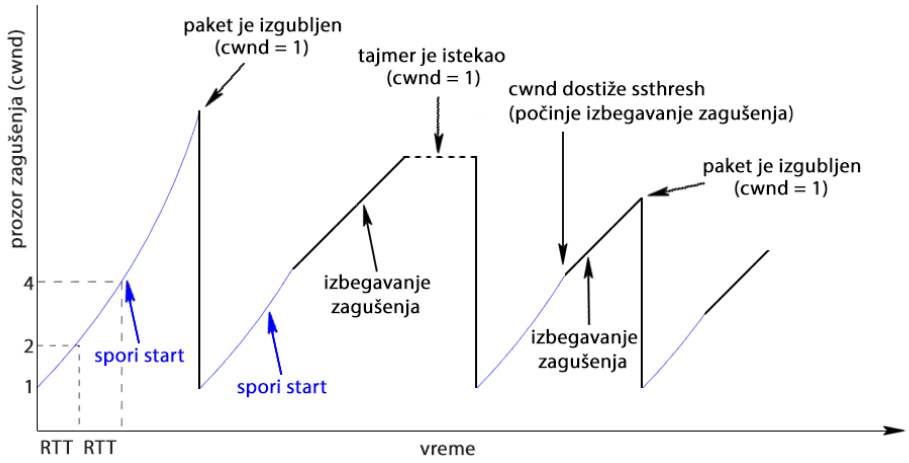
TCP je uvek ili u fazi sporog starta ili u izbegavanju zagušenja. Slika 20 prikazuje konačni automat algoritma Tahoe, sa akcijama koje se izvršavaju.



Slika 20: Konačni automat ponašanja Tahoe algoritma sa akcijama

Uvodi se nova promenljiva, koja se naziva *ssthresh* (od eng. Slow Start Threshold). Nju održava TCP i ona predstavlja gornju granicu za *cwnd* u sporom startu. Kada se ta granica pređe, počinje faza izbegavanja zagušenja. Tahoe kao signal da se paket izgubio koristi troduplu potvrdu (tri puta pristiglu istu potvrdu; koristi se i skraćenica 3dup ACK). Kako TCP koristi kumulativne potvrde, ako više puta stigne potvrda sa istim brojem, to obično znači da se paket sa tim brojem izgubio. Ako se to desi, ili tajmer za paket istekne, ostaje se u fazi sporog starta, s tim što se dodeljuje *ssthresh* = *cwnd*/2 i *cwnd* se resetuje na 1. Na samom početku *ssthresh* se postavlja na neku veoma veliku vrednost, primoravajući tako algoritam na gubitak paketa i postavljanje adekvatnije vrednosti za *ssthresh* (*cwnd*/2) na samom početku.

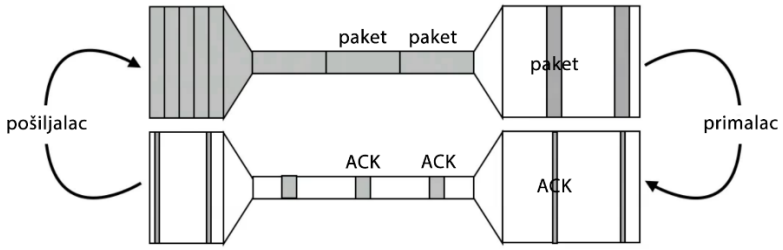
U fazi izbegavanja zagušenja TCP ostaje sve dok se paket ne izgubi (trodupla potvrda ili kada tajmer istekne). Kada se to desi, akcija je ista kao i malopre, *ssthresh* = *cwnd*/2 i *cwnd* = 1, a prelazi se u stanje sporog starta.



Slika 21: Tahoe - promena prozora zagušenja; treba primetiti da se *cwnd* posle bilo kakvog gubitka resetuje na 1; naizmenično menjanje sporog starta i izbegavanja zagušenja liči na AIMD

3.2.4 Samotempiranje

Tahoe je uveo samotempiranje [15] kao proces pri slanju koji vremenski jednako odvaja pakete jedan od drugog i time omogućava ravnomernu brzinu slanja.

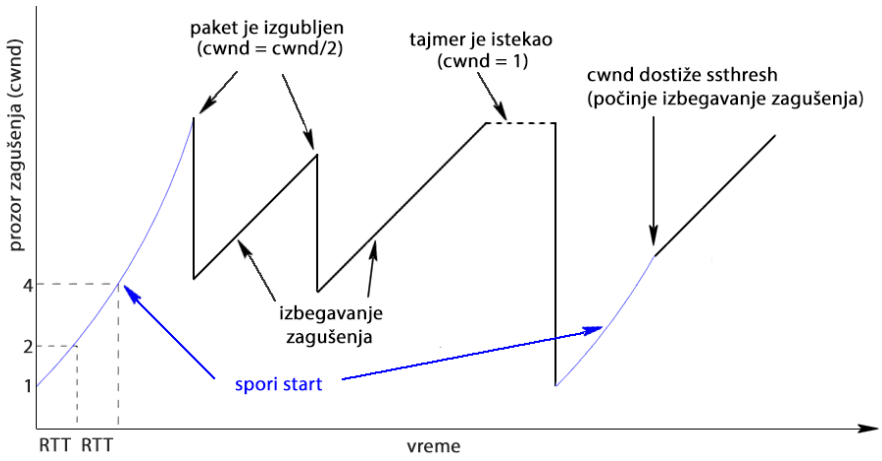


Slika 22: Samotempiranje – paketi su ravnomerno vremenski raspoređeni

Na slici 22 prikazana su oba smera, pošiljalac-primalac i obrnuto. Ka primaocu „idu“ paketi sa podacima, a obrnuto potvrde za te pakete. Sužen deo u sredini predstavlja usko grlo, neko ograničenje brzine, pa su paketi tu razvučeni, i potrebno im je više vremena da prođu. Količina podataka može se predstaviti površinom svakog paketa na slici, koja ostaje uvek ista, samo se „razvuče“ kada se naiđe na usko grlo. Primalac za svaki paket šalje potvrdu, i one se šalju u jednakim vremenskim intervalima (u zavisnosti od toga kako paketi stižu). Sa druge strane, svaki novi ACK je signal da se može poslati novi paket, pa su i oni jednako vremenski razdvojeni. Ovim se postiže da se prozor paketa ne šalje naglo, što bi moglo da optereti mrežu, već ravnomerno, paket po paket, dajući im tako najbolju šansu da stigu do odredišta.

3.3 RENO I NEWRENO

Tahoe je rešio problem kada ima zagušenja, ali se primetilo da transfer može biti efikasniji. Reno je algoritam koji je uveo dva poboljšanja. Na slici 23 predstavljena je simulacija ponašanja algoritma.



Slika 23: Reno – promena prozora zagušenja (primetiti da je faza izbegavanja zagušenja AIMD)

Uvidelo se da je nepotrebno resetovati *cwnd* na jedan nakon troduple potvrde, jer je velika verovatnoća da je brzina slanja u tom trenutku tek nešto veća od optimalne. Reno ovde postavlja *cwnd* = *cwnd*/2 i ostaje u fazi izbegavanja zagušenja. Ovo zovemo brz oporavak (eng. Fast Recovery) [18]. Drugo poboljšanje je da se, kada troupla potvrda stigne, ne čeka na istek tajmera, već se odmah šalje paket koji je nestao. Ovo je brzo ponovno slanje (eng. Fast Retransmit) [18]. Sve ostalo je isto kao kod Tahoe algoritma.

NewReno [19] dodaje jedno poboljšanje u odnosu na Reno. Tokom faze brzog oporavka, Reno će poslati paket koji fali, ali će onda stati da sačeka potvrdu, jer mehanizam kliznog prozora ne dozvoljava da se isti „pomeri“ dalje, zbog paketa koji nedostaje. NewReno koristi tehniku koju zovemo naduvavanje prozora (eng. Window Inflation). Ona se sastoji u tome, da tokom brzog oporavka dozvoli prozoru da polako raste (za jedan na svaku novu, dupliranu, potvrdu koja signalizira pomenuti izgubljeni paket), kako bi omogućio novim paketima da odlaze, bez čekanja. Na kraju, kada potvrda za izgubljeni paket stigne, prozor se vraća na veličinu koju je imao pre naduvavanja. Do ove tehnike se došlo tako što se primetilo da kada se paket izgubi, obično je brzina prekoračena, ali je sve u redu sa mrežom i možemo nastaviti dalje slanje, nema potrebe da TCP stane i čeka potvrdu. Ova tehnika nije osobena samo za NewReno, već je podrazumevano koriste i drugi algoritmi. Često se ovaj algoritam naziva samo Reno, pa će se i u daljem tekstu, kada se koristi ime Reno, misliti na NewReno.

4 PROBLEMI SA KLASIČNIM ALGORITMIMA

Reno se uglavnom naziva „standardni TCP“ i on je najčešće korišćen, obično uz neke modifikacije. On je uglavnom dobar i većina „kućnih“ korisnika se nikada neće ni zapitati koji algoritam koristi njihov operativni sistem. Reno uspeva da reši problem zagušenja u jednom smeru, kada zagušenja ima. Međutim, veze sa velikim propusnim opsegom (već 10Mb/s i više) i velikim vremenom obilaska (50ms i više) (eng. Big Fat Pipes) uzrokuju da ovakav algoritam bude jako neefikasan u iskorišćavanju potencijalnog kapaciteta. Kako ovakve veze postaju sve pristupačnije, čak i za tzv. kućne korisnike, raste i potreba da se ovaj problem reši. Tokom godina predložena su mnoga rešenja, od kojih će nekoliko biti predstavljeno u narednom poglavlju. Dalji tekst opisuje dva problema, veliki proizvod propusnog opega i vremena obilaska (eng. Bandwidth-Delay Product – BDP), i nefer alokaciju prema vremenu obilaska (eng. RTT Fairness).

4.1 VELIKI BDP

BDP, proizvod propusnog opega i vremena obilaska, meri koliko podataka treba slati (veličina kliznog prozora), da bi se kapacitet veze maksimalno iskoristio. Što su veći kapacitet i RTT, veći je BDP. Npr., zamislimo brzu vezu od 10Gb/s koja spaja tačke između dva kontinenta (npr. Evropa i Amerika), sa vremenom obilaska od 100ms. Izračunajmo njen BDP. BDP = 10Gb/s * 0,1s = 1Gb (~120MB). Proizvod propusnog opega i vremena obilaska ovde je ogroman. Prozor zagušenja bi morao da naraste do 120MB kako bi se iskoristio potencijal veze. Standardni algoritam, uvećavajući prozor za jedan na svaki RTT (100ms ovde), dostigao bi odgovarajuću veličinu prozora, odnosto maksimalnu brzinu, za oko sat vremena [20]. I to bi bio idealan slučaj, gde se pretpostavlja da nijedan paket neće biti izgubljen (jedan izgubljen paket na ~$2{,}5*10^9$ poslatih). Ovo je daleko od onoga što je moguće današnjom tehnologijom [21], [22]. Čak i ako se zamisli da je ovako mali procenat izgubljenih paketa moguć, sat vremena je previše čekanja da se dostigne maksimalna brzina. Dalje, kada (ako) se ona dostigne, jedan paket će biti izgubljen, zbog prekoračenja maksimalne brzine, i prozor će se prepoloviti. Biće potrebno pola sata da prozor ponovo naraste, i tako dalje.

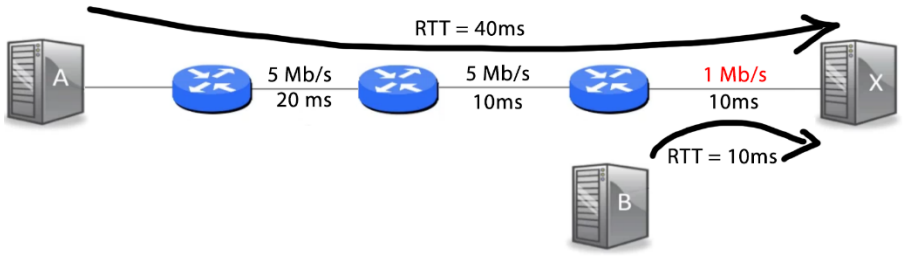
Opisan primer podrazumeva idealnu situaciju gde nema zagušenja i gde ne postoje drugi tokovi koji se takmiče za kapacitet. U stvarnosti, uvek postoji puno tokova koji međusobno utiču jedni na druge. TCP u svakom trenutku treba dinamički da se ponaša i da uzima svoj fer deo najbrže moguće (jer ga u suprotnom neko drugi uzima), a sa druge strane da uspori na znak zagušenja. Sporo reagovanje (reda veličine sata!), uzrokuje da prenos podataka ovakvim algoritmom na vezama sa velikim proizvodom propusnog opega i vremena obilaska bude ne samo neefikasan, već neizvodljiv [23].

4.2 NEFER ALOKACIJA PREMA VREMENU OBILASKA

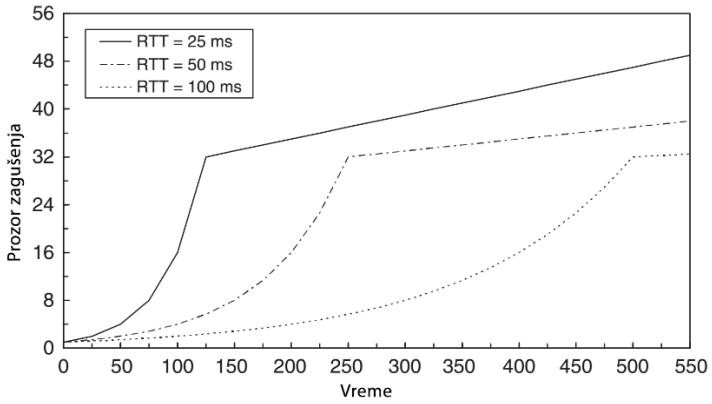
Generalna ideja AIMD šeme i njene implementacije u TCP-u je da se prozor zagušenja uvećava za jedan na svaki RTT, a smanjuje se za polovinu svoje veličine na znak zagušenja

(izgubljeni paket). Multiplikativno smanjenje je u redu, ali aditivno uvećanje ima manu. Funkcija uvećanja jeste linearna, ali nema uvek istu brzinu rasta i zavisna je od vremena obilaska [9], [10], [11]. RTT ima analogan uticaj i na spori start, samo je tu rast eksponencijalan umesto linearnog.

Na slici 24 dat je primer dva računara koji dele usko grlo, gde jedan ima veoma veće vreme obilaska od drugog i zato je u nepovoljnoj situaciji. Što je veći RTT, funkcija aditivnog uvećanja prozora raste sporije (jedan paket na svaki RTT), a time i brzina, s obzirom da je ona direktno proporcionalna veličini prozora.

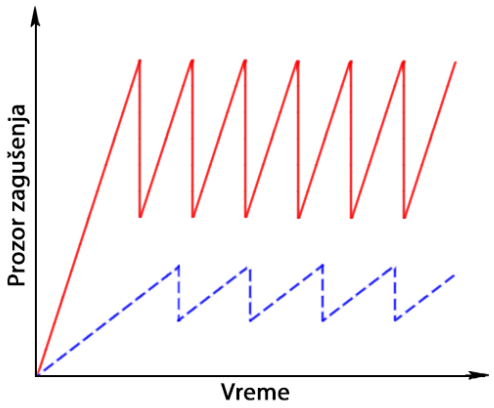


Slika 24: Računari A i B šalju podatke računaru X i dele usko grlo; putanja računara A do X je mnogo duža i ima veće kašnjenje (40ms) u odnosu na putanju B-X (10ms); poželjno je da se kapacitet uskog grla deli jednako, ali je A u nepovoljnoj situaciji zbog većeg RTT-a, i standardni TCP mu neće dodeliti fer deo



Slika 25: Simulacija sporog starta i aditivnog uvećanja pri različitim vremenima obilaska

Moglo bi se pretpostaviti da, zbog multiplikativnog smanjenja, tokovi vremenom ipak konvergiraju ka svojim fer delovima kapaciteta. Međutim, kada se RTT tokova veoma razlikuju, tok sa manjim kašnjenjem će uvećati svoj prozor dosta brže u odnosu na tok sa većim u fazi izbegavanja zagušenja. Ispostavi se da multiplikativno smanjenje, iako smanji prozor toka sa manjim kašnjenjem više, ne rešava problem, jer će taj tok ugrabiti veći deo kapaciteta ponovo kada uđe u fazu aditivnog uvećanja.



Slika 26: Dva toka se takmiče; tok označen isprekidanom linijom ima četiri puta veće vreme obilaska od drugog toka; iako multiplikativno umanjenje više utiče na tok sa manjim kašnjenjem, kako je i planirano, aditivno uvećanje uvek dozvoljava drugom toku da brzo uveća svoj prozor i uzme veći deo kapaciteta

Ovaj problem postoji čak i u mnogim modernim algoritmima i nije ga jednostavno rešiti. Veliki RTT generalno znači da će TCP sporije reagovati, jer će povratna informacija više kasniti. Algoritmi koji rešavaju problem nefer alokacije prema RTT-u, obično pokušavaju da postave pseudo vrednost kašnjenja toka na željenu (videti 5.4 Hybla).

5 MODERNA REŠENJA I ALGORITMI

U poslednje dve decenije problem zagušenja je bio prilično interesantan za rešavanje, pa je iz toga proizišlo puno algoritama i nekoliko različitih grupa u koje ih možemo smestiti. Prva, i osnovna, jeste grupa reaktivnih (eng. Reactive) algoritama čiji je znak zagušenja izgubljeni paket, na koji oni reaguju na određeni način (otuda i ime). Ovde „upadaju“ svi klasični algoritmi, ali i većina modernih, od kojih će biti opisani BIC, CUBIC i Hybla. Drugu grupu čine tzv. proaktivni (eng. Proactive, Delay-based) algoritmi. Znak za nadolazeće zagušenje je uvećanje vremena obilaska, jer paketi duže ostaju u baferima rutera. Glavni predstavnik ove grupe, koji će biti opisan, jeste Vegas. Treća grupa bi bila „reaktivna sa procenom propusnog opsega“, a postoje i kombinacije pomenutih. Npr. kombinacija reaktivnog i proaktivnog algoritma, kao što je Illinois, što može biti veoma dobar pristup, kao što ćemo videti u daljem tekstu.

5.1 SACK I FACK

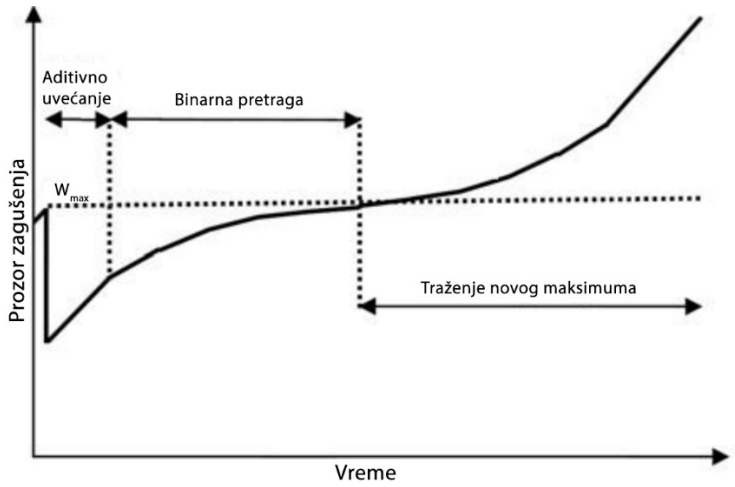
Tehnika kliznog prozora, zajedno sa kumulativnom potvrdom veoma je konzervativan pristup, posebno ako se odjednom izgubi više paketa iz istog prozora. Kumulativna potvrda je ograničena informacija o gubitku paketa, i TCP saznaje samo za jedan izgubljen paket („najlevlji“) za jedno vreme obilaska, i ne zna da li su ostali paketi stigli ili ne. SACK (eng. Selective Acknowledgement) [24] je opcija koja rešava ovaj problem. Ona omogućava da se pošiljaocu dostavi kompletnija informacija o paketima koji su uredno stigli i o eventualnim „rupama“ (nedospelim paketima), pa je TCP u mogućnosti da odjednom pošalje sve izgubljene pakete. FACK (Forward Acknowledgement) [25] dalje unapređuje SACK. Moderniji operativni sistemi imaju obe opcije podrazumevano uključene.

5.2 BIC

BIC [26] algoritam vidi problem zagušenja kao problem pretrage, gde „sistem“ daje „da – ne“ odgovor kroz gubitak paketa na pitanje da li je trenutni prozor veći od mogućnosti mreže da ga podrži. Trenutni minimum za veličinu prozora se može odrediti kao vrednost pri kojoj nema gubitaka paketa, i ako je maksimum poznat, može se koristiti binarna pretraga kako bi se našla optimalna vrednost. Argument za ovu strategiju je da ako se gubitak paketa desi oko maksimalne vrednosti, a oko minimalne ne, onda je optimum negde između ta dva.

Slika 27 predstavlja funkciju rasta prozora zagušenja BIC algoritma. Kada se paket izgubi, BIC smanjuje prozor za multiplikativni faktor β. Veličina prozora neposredno pre smanjenja je W_{max}, a veličina nakon W_{min}. BIC izvršava binarnu pretragu, sa levim krajem W_{min}, i desnim W_{max}. Kako je paket izgubljen pri veličini prozora W_{max}, veličina prozora koju mreža može da podnese je verovatno negde između te dve vrednosti. Pošto „skok“ na „srednju vrednost“ (binarnom pretragom) može bit preveliko uvećanje prozora za jedan RTT, ako je razlika između minimuma i srednje tačke veća od zadate konstante, nazvane S_{max}, BIC uvećava svoj prozor za S_{max}. Ako se sa novim prozorom ne desi gubitak paketa, ta veličina postaje novi minimum. Ako se paket izgubi,

to je novi maksimum. Ovo se nastavlja sve dok uvećanje prozora ne postanje manje od konstante S_{min}, kada se prozoru jednostavno dodeljuje vrednost maksimuma. Funkcija veličine prozora nakon izgubljenog paketa će najverovatnije ličiti na onu sa slike 27, prvo linearna (uvećanja za S_{max}), pa nakon toga logaritamska (binarna pretraga).



Slika 27: BIC funkcija prozora – približavanje starom i traženje novog maksimuma

Ako prozor naraste preko vrednosti maksimuma, BIC algoritam to interpretira kao da je optimalna vrednost za prozor veća (dobijen je veći kapacitet), i treba naći novi maksimum. Funkcija je ovde inverzna u odnosu na onu koja dostiže maksimum, opisanu malopre, i u obrnutom redosledu. Prvo se prozor uvećava eksponencijalno (inverzno logaritmu), da bi se brzo našla nova vrednost, a potom, kada uvećanje postane preveliko (veće od zadate konstante S_{max}), aditivno. BIC funkcija je data pseudokodom.

```
BIC(Smax, Smin)
begin
Wc = 1; // trenutna velicina prozora

MAX_PROBING:
  if("ACK") then Wc = Wc + 1; // eksp. uvecanje
  else if("packet loss") then goto PACKET_LOSS;
  else goto MAX_PROBING; // ponovo

PACKET_LOSS:
  // pocni aditivno uvecanje
  Wmax = Wc;
  Wc = Wmax / 2;
  goto ADDITIVE_INCREASE;

ADDITIVE_INCREASE:
  if("ACK") then
    if(Wmax - Wc < Smin) then
      goto MAX_PROBING;
    Wt = (Wc + Wmax) / 2; // sacuvaj Wt
    if(Wt – Wc > Smax) then
      Wc = Wc + Smax;
      goto ADDITIVE_INCREASE; // ponovo
    else goto BINARY_SEARCH;
  else if("packet loss") then
        goto PACKET_LOSS;
      else goto ADDITIVE_INCREASE;

BINARY_SEARCH:
  if("ACK") then
    Wc = Wt; // Wt je sacuvan ranije
    if(Wmax - Wc < Smin) then
      goto MAX_PROBING;
    else
      Wt = (Wc + Wmax) / 2;
      goto BINARY_SEARCH; // ponovo
  else if("packet loss") goto PACKET_LOSS;
       else goto BINARY_SEARCH; // ponovo
end.
```

5.3 CUBIC

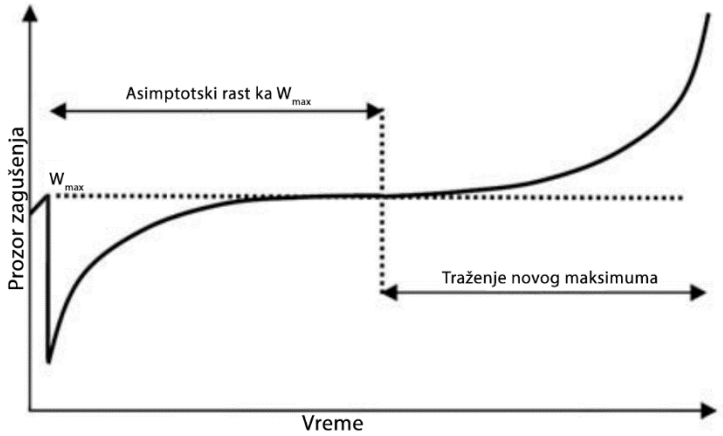
Iako BIC postiže dobru skalabilnost i stabilnost u okruženjima sa velikim kapacitetom mreže, njegovi autori su ocenili da funkcija rasta prozora može biti previše agresivna na vezama sa malim vremenom obilaska i malim propusnim opsegom, pa su potražili novu funkciju koja bi uklonila nedostatak a zadržala prednosti. Predložena revizija algoritma zove se CUBIC [27].

CUBIC, kao što se iz imena može naslutiti, se zasniva na kubnoj funkciji koja reguliše rast prozora zagušenja. Preciznije, prozor je određen funkcijom

$$W_{cubic} = C(t-K)^3 + W_{\max} \tag{5}$$

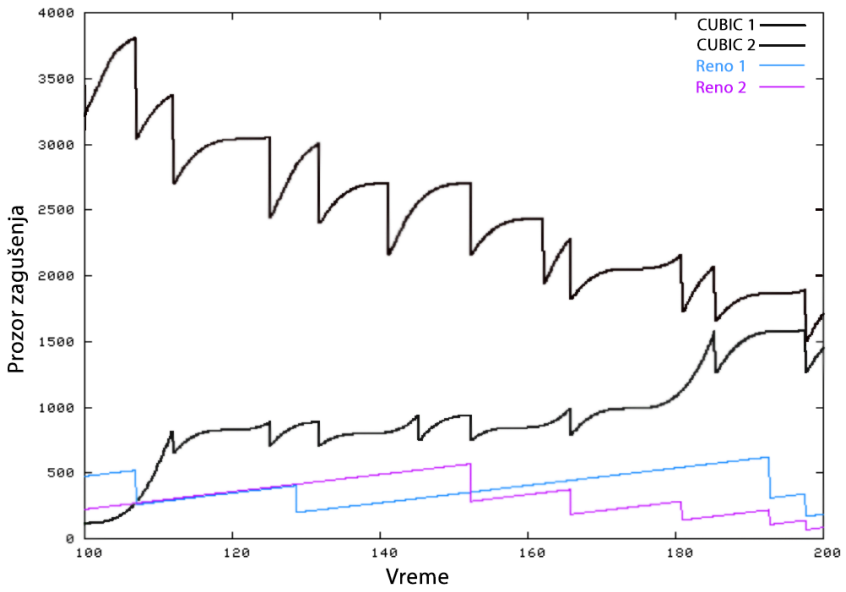
gde je *C* faktor skaliranja, *t* je vreme proteklo od poslednje redukcije prozora, W_{max} je veličina prozora neposredno pre poslednje redukcije, a $K = \sqrt[3]{W_{\max}\beta/C}$, gde je *β* multiplikativni faktor umanjenja.

Slika 28 predstavlja funkciju prozora nakon izgubljenog paketa, tj. smanjenja prozora. Isprva prozor raste veoma brzo, ali usporava kako se približava W_{max}. Blizu W_{max}, uvećanje je skoro nula. Kada se ta tačka pređe, CUBIC, kao i BIC, počinje pretragu za novim maksimumom, prvo lagano, ubrzavajući rast kako se odmiče od W_{max}. Spori rast oko W_{max} povećava stabilnost algoritma, dok kasniji brži rast omogućava skalabilnost.



Slika 28: CUBIC funkcija prozora - približavanje starom i traženje novog maksimuma

Kubna funkcija omogućava fer alokaciju između tokova koji koriste isti protokol. Kako oba izvršavaju redukciju prozora za isti faktor, *β*, veći prozor će se umanjiti više, a funkcija rasta, (1), osigurava da tok sa većim W_{max} raste sporije, jer je *K* veće ako je W_{max} veće. Funkcija takođe nudi dobru fer alokaciju prema vremenu obilaska, jer rast prozora uglavnom zavisi od *t*, pošto će ono biti isto za sve tokove nakon izgubljenog paketa, ako pretpostavimo da ga gube istovremeno.



Slika 29: Simulacija ponašanja dva CUBIC i dva Reno toka koji se međusobno takmiče, gde je C = 0,4 i β = 0,8 za CUBIC; primetiti da dva CUBIC toka vremenom konvergiraju, i da rast prozora jeste kubna funkcija, kao što je i opisano ranije; zahvaljujući većoj skalabilnosti, CUBIC tokovi ostvaruju veliku prednost u odnosu na Reno

5.4 HYBLA

Osnova algoritma Hybla [28] je da toku sa velikim vremenom obilaska obezbedi istu brzinu slanja u odnosu na brzi referentni tok. Zamišljen je da se koristi za satelitske ili druge bežične veze, ali se njegove prednosti mogu efikasno koristiti i pri žičnim koje imaju veliki BDP. Dalji tekst se fokusira na glavno svojstvo algoritma, eliminaciju zavisnosti veličine prozora od vremena obilaska.

Neka je *W(t)* veličina prozora zagušenja standardnog TCP algoritma u vremenu *t*, koje je na početku 0, i neka je t_γ vreme kada prozor dostiže *ssthresh* vrednost (trenutak kada se iz faze sporog starta, obeleženo sa SS, prelazi u izbegavanje zagušenja, obeleženo sa CA), *γ*. *W(t)* je dato sledećom jednačinom

$$W(t) = \begin{cases} 2^{t/\mathrm{RTT}}, & 0 \leqslant t < t_\gamma, \quad \mathrm{SS} \\ \dfrac{t - t_\gamma}{\mathrm{RTT}} + \gamma, & t \geqslant t_\gamma, \quad \mathrm{CA} \end{cases} \tag{6}$$

gde je RTT vreme obilaska toka (RTT i *t* se mere istim jedinicama vremena, npr. sekunda). Veličina prozora u bajtovima se može dobiti množenjem *W(t)* sa veličinom maksimalnog segmenta koji se šalje. Iz (6) je evidentno da veličina prozora zavisi od vremena obilaska.

Neka je *B(t)* „brzina“ toka, tj. broj paketa po sekundi koji se šalju u trenutku *t*,

$$B(t) = W(t)/\mathrm{RTT} \tag{7}$$

(jer se *W(t)* paketa se pošalje na svaki RTT), i neka $T_d(t)$ označava količinu poslatih podataka od početka (od t=0). Tada, iz (6) i (7), sledi

$$T_d(t) = \int_0^t B(\tau)\,\mathrm{d}\tau = \begin{cases} \dfrac{2^{t/\mathrm{RTT}} - 1}{\ln(2)}, & 0 \leqslant t < t_\gamma, \quad \mathrm{SS} \\ \dfrac{\gamma - 1}{\ln(2)} + \dfrac{(t - t_\gamma)^2}{2\mathrm{RTT}^2} + \dfrac{\gamma(t - t_\gamma)}{\mathrm{RTT}}, & t \geqslant t_\gamma, \quad \mathrm{CA} \end{cases} \tag{8}$$

što, naravno, takođe zavisi od vremena obilaska.

Jednačina (7) sugeriše da je moguće vreme obilaska „izbaciti iz igre“, tako što bi se *W(t)* učinilo nezavisno od RTT-a, i nadomeštanjem deljenja sa RTT-om. U tom cilju, Hybla uvodi normalizovano vreme obilaska, ρ, dato sa

$$\rho = \mathrm{RTT}/\mathrm{RTT}_0 \tag{9}$$

gde je RTT_0 referentno vreme obilaska (poželjno vreme, obično vrlo malo, npr. 25ms). Prvo se vreme, t, množi sa ρ, čineći time *W(t)* nezavisno od RTT-a, a onda se rezultujući prozor množi sa ρ, kako bi se *B(t)* učinio takođe nezavisnim. Dobija se

$$W^{\mathrm{H}}(t) = \begin{cases} \rho 2^{\rho\, t/\mathrm{RTT}}, & 0 \leqslant t < t_{\gamma,0}, \quad \mathrm{SS} \\ \rho\left[\rho\,\dfrac{t - t_{\gamma,0}}{\mathrm{RTT}} + \gamma\right], & t \geqslant t_{\gamma,0}, \quad \mathrm{CA} \end{cases} \tag{10}$$

gde je sada $t_{\gamma,0}$ simbol za trenutak kada prozor dostiže vrednost $\rho\gamma$. Iz (7) i (10) dobija se

$$B^{\mathrm{H}}(t) = \begin{cases} \dfrac{2^{t/\mathrm{RTT}_0}}{\mathrm{RTT}_0}, & 0 \leqslant t < t_{\gamma,0}, \quad \mathrm{SS} \\ \dfrac{1}{\mathrm{RTT}_0}\left[\dfrac{t - t_{\gamma,0}}{\mathrm{RTT}_0} + \gamma\right], & t \geqslant t_{\gamma,0}, \quad \mathrm{CA} \end{cases} \tag{11}$$

što je nezavisno od vremena obilaska i jednako brzini (fiktivnog) toka sa referentnim vremenom obilaska RTT_0. Iz (11) jednostavno je izračunati količinu prenetih podataka,

$$T_d^{\mathrm{H}}(t) = \int_0^t B^{\mathrm{H}}(\tau)\,\mathrm{d}\tau = \begin{cases} \dfrac{2^{t/\mathrm{RTT}_0} - 1}{\ln(2)}, & 0 \leqslant t < t_{\gamma,0}, \quad \mathrm{SS} \\ \dfrac{\gamma - 1}{\ln(2)} + \dfrac{(t - t_{\gamma,0})^2}{2\mathrm{RTT}_0^2} + \dfrac{\gamma(t - t_{\gamma,0})}{\mathrm{RTT}_0}, & t \geqslant t_{\gamma,0}, \quad \mathrm{CA} \end{cases} \tag{12}$$

odakle je očigledno da količina prenetih podataka više ne zavisi od vremena obilaska.

Što se tiče implementacije Hybla algoritma, ona se može postići modifikovanjem funkcije rasta prozora zagušenja na sledeći način.

$$W_{i+1}^{\mathrm{H}} = \begin{cases} W_i^{\mathrm{H}} + 2^{\rho} - 1, & \mathrm{SS} \\ W_i^{\mathrm{H}} + \rho / W_i^{\mathrm{H}}, & \mathrm{CA} \end{cases} \tag{13}$$

Minimalna vrednost za ρ je 1, pa se Hybla ponaša kao standardni TCP algoritam kada je RTT malo ($RTT < RTT_0$). Dodatno, inicijalna vrednost prozora i originalna *ssthresh* vrednost moraju biti pomnožene sa ρ, kao i veličina bafera za prenos, kako bi podržao veće nalete podataka.

5.5 VEGAS

Vegas [29] je glavni predstavnik proaktivne grupe algoritama. Za razliku od standardnog načina detektovanja zagušenja, gubitka paketa, Vegas meri vreme obilaska za svaki paket, tj. vreme od slanja paketa do primanja potvrde za njega. Prvo se računa „bazni RTT“, i to je prosečno i očekivano vreme obilaska paketa datog toka. Ako se RTT povećava u odnosu na bazni, Vegas pretpostavlja da zagušenje počinje, jer se paketi zadržavaju duže od očekivanog u baferima usput. To je znak da prozor zagušenja treba smanjiti. Sa druge strane, ako je RTT manji od baznog, prozoru se dopušta da poraste. Da bi se omogućilo preciznije merenje vremena obilaska, Vegas koristi vremenski pečat (eng. Timestamp) kao TCP opciju i finije RTT merenje.

Ovaj algoritam zavisi od jake korelacije između vremena obilaska i zagušenja u mreži. Zbog toga ostvaruje 40%-70% bolje rezultate od standardnog TCP algoritma u homogenim mrežama, gde je ta korelacija jaka [30]. Međutim, u heterogenim mrežama, kao što je Internet, ta korelacija je pod znakom pitanja, i istraživanja pokazuju da je manja od 0,18 u proseku [31], što je nedovoljno da bi se uvećanje RTT-a koristio kao jedini znak zagušenja.

Vegas, kao posledica proaktivnog načina detektovanja zagušenja, često uzima manje od svog fer dela kapaciteta kada se takmiči sa tokovima koji koriste reaktivne algoritme. Razlog za to je što Vegas reaguje, smanjuje svoj prozor, pre nego što zagušenje počne, pa ostali tokovi tako dobijaju priliku dalje da uvećaju svoje. Takoreći, proaktivni algoritmi odustaju od trke za kapacitet ranije, što može da ima loše posledice po performanse.

5.6 ILLINOIS

Illinois [32] koristi kombinovani pristup. Gubitak paketa je primarni signal za smanjenje prozora, ili njegovo uvećanje (kada gubitka nema), a razlika „baznog RTT-a“ i trenutnog se koristi kao sekundarni signal koji određuje koliko će se prozor smanjiti, tj. uvećati. Time Illinois postiže cilj algoritma kao što je Vegas, da se približi maksimumu brzine i da tu ostane što duže (za razliku od Reno-a, koji bi brzo prekoračio maksimum i smanjio prozor), a pritom izbegva sve njegove slabosti koje donosi čist pristup zasnovan na kašnjenju.

Slično kao i standardni TCP, Illinois koristi sledeću formulu za izračunavanje vrednosti prozora

$$W_{i+1} = \begin{cases} W_i + \alpha/W_i, & \text{na ACK} \\ W_i - \beta W_i, & \text{na gubitak paketa} \end{cases} \tag{14}$$

s tim što α i β nisu konstante. Na osnovu trenutnog vremena obilaksa u odnosu na bazno, α i β se računaju sledećim formulama.

$$\alpha = f_1(d_c) = \begin{cases} \alpha_{max}, & d_c \le d_1 \\ \dfrac{k_1}{k_2 + d_c}, & \text{inače} \end{cases}$$

$$\beta = f_2(d_c) = \begin{cases} \beta_{min}, & d_c \le d_2 \\ k_3 + k_4 d_c, & d_2 < d_c < d_3 \\ \beta_{max}, & \text{inače} \end{cases} \tag{15}$$

tako da važi $\alpha_{min} \le \alpha \le \alpha_{max}$ i $\beta_{min} \le \beta \le \beta_{max}$, gde su

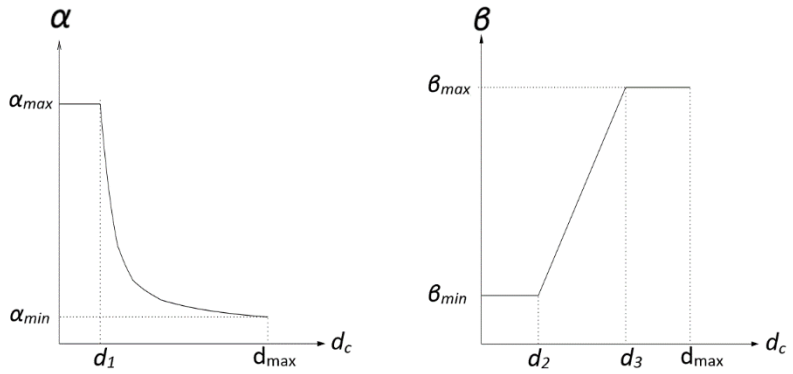
$$\alpha_{min} = \frac{3}{10}, \alpha_{max} = 10$$
$$\beta_{min} = \frac{1}{8}, \beta_{max} = \frac{1}{2} \tag{16}$$

d_c je trenutno izmereni RTT, d_1, d_2 i d_3 se izračunavaju u zavisnosti od baznog vremena obilaska, a k_1, k_2, k_3 i k_4 se biraju tako da f_1 i f_2 budu neprekidne, na ovaj način

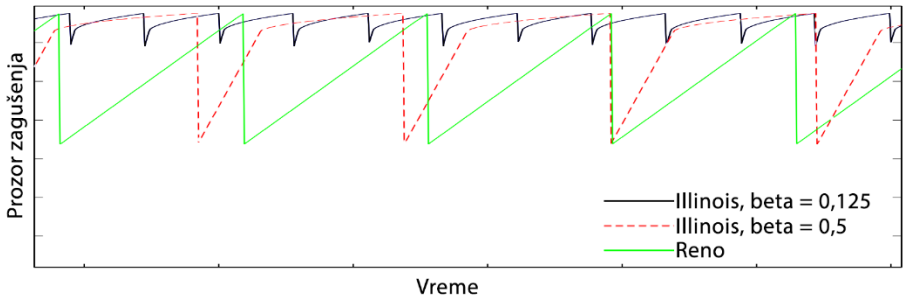
$$k_1 = \frac{(d_{max} - d_1)\alpha_{min}\alpha_{max}}{\alpha_{max} - \alpha_{min}}, \quad k_2 = \frac{(d_{max} - d_1)\alpha_{min}}{\alpha_{max} - \alpha_{min}} - d_1$$
$$k_3 = \frac{d_3\beta_{min} - d_2\beta_{max}}{d_3 - d_2}, \quad k_4 = \frac{\beta_{max} - \beta_{min}}{d_3 - d_2} \tag{17}$$

gde je d_{max} maksimalno vreme kašnjenja.

Iz (15), (16) i (17) sledi da α opada konveksno kako se RTT povećava, pa prozor raste konkavno kako se približava maksimalnoj vrednosti, a β raste linearno (slika 30). Ovo omogućava da se veličina prozora duže zadrži blizu maksimalne, pre nego što se desi zagušenje, i da se istovremeno donekle proceni stepen zagušenja (prema vremenu obilaska), pa prema tome odredi koliko će se prozor smanjiti. Iz toga direktno sledi bolja iskorišćenost kapaciteta, što se može ilustrovati slikom 31.



Slika 30: Prikaz funkcija za α i β u zavisnosti od trenutnog RTT-a, d_c; α konveksno opada, a β linearno raste sa rastom vremena obilaska

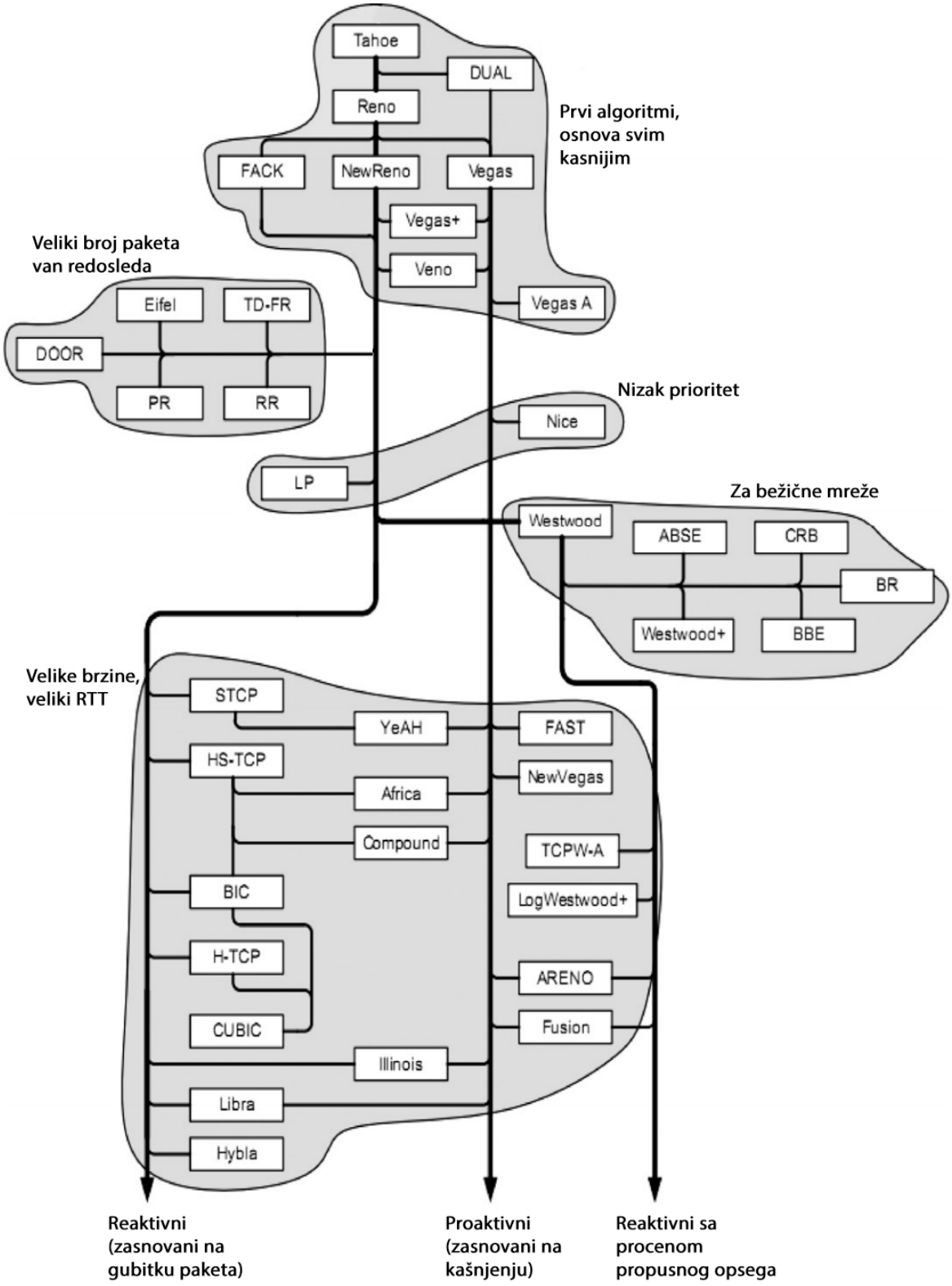


Slika 31: Simulacija rasta prozora Illinois algoritma sa različitim vrednostima za β (0,125 i 0,5) i standardno, Reno, ponašanje; primetiti da konkavnost prozora blizu maksimuma omogućava da se to stanje duže održi, pre nego se prozor smanji usled znaka za zagušenje

5.7 DRUGI

Glavna tema rada su veze velikog kapaciteta sa velikim kašnjenjem i algoritmi koji rešavaju probleme koji proističu iz njih. Postoje i druge grupe algoritama koji rešavaju drugačije probleme, i zaslužuju da budu pomenuti. TCP Westwood [33] je predstavnik algoritama za bežične mreže. One zahtevaju drugačiji pristup, jer je gubitak paketa kada nema zagušenja daleko veći u odnosu na žične veze, pa se ne može smatrati sigurnim znakom zagušenja. LP [34] i Nice [35] su algoritmi niskog prioriteta, koji su osmišljeni tako da uzimaju samo slobodni kapacitet i da se nikada ne takmiče sa drugim tokovima. TCP DOOR [36] je napravljen tako da bude jako robusan na pojavu velikih količina paketa van redosleda.

Slika 32 prikazuje stablo razvoja algoritama, podeljenih u grupe prema problemima koje rešavaju. Stablo je predstavljeno hronološki, sa međusobnim uticajima, gde je „koren“ na vrhu.



Slika 32: Stablo razvoja algoritama za kontrolu zagušenja; poređani su hronološki i podeljeni u grupe prema problemima koje rešavaju; strelice označavaju uticaj prethodnih i tehnike koje koriste

6 EKSPERIMENTALNA ANALIZA

U prethodnim poglavljima opisani su problem zagušenja i algoritmi, klasični i moderni, koji se koriste kako bi se on rešio i ostvario efikasan protok. Da bi rad bio potpun, izvršen je određen broj testova, što u kontrolisanim, sintetičkim uslovima, što na javnoj mreži, kako bi se izmerile i prezentovale performanse ranije opisanih algoritama. Testirani su Reno, BIC, CUBIC, Hybla, Vegas i Illinois. SACK i FACK opcije su bile uključene za svaki algoritam.

6.1 METODOLOGIJA

Prva grupa testova je obavljena u lokalnoj mreži sa dva računara, povezana direktno. Računari su Intel Pentium IV procesor na 1,8GHz sa 2GB RAM-a, koji rade pod Linux Ubuntu 12.04 operativnim sistemom. Konekcija između njih je bila ograničena na 100Mb/s. Testovi su osmišljeni tako da pokažu uticaj vremena obilaska i gubitka paketa na performanse svakog algoritma. Da bi se to implementiralo, veštački je proizvedeno kašnjenje, kao i nasumični gubici paketa u određenoj količini. Za ovo je korišćen program tc [37].

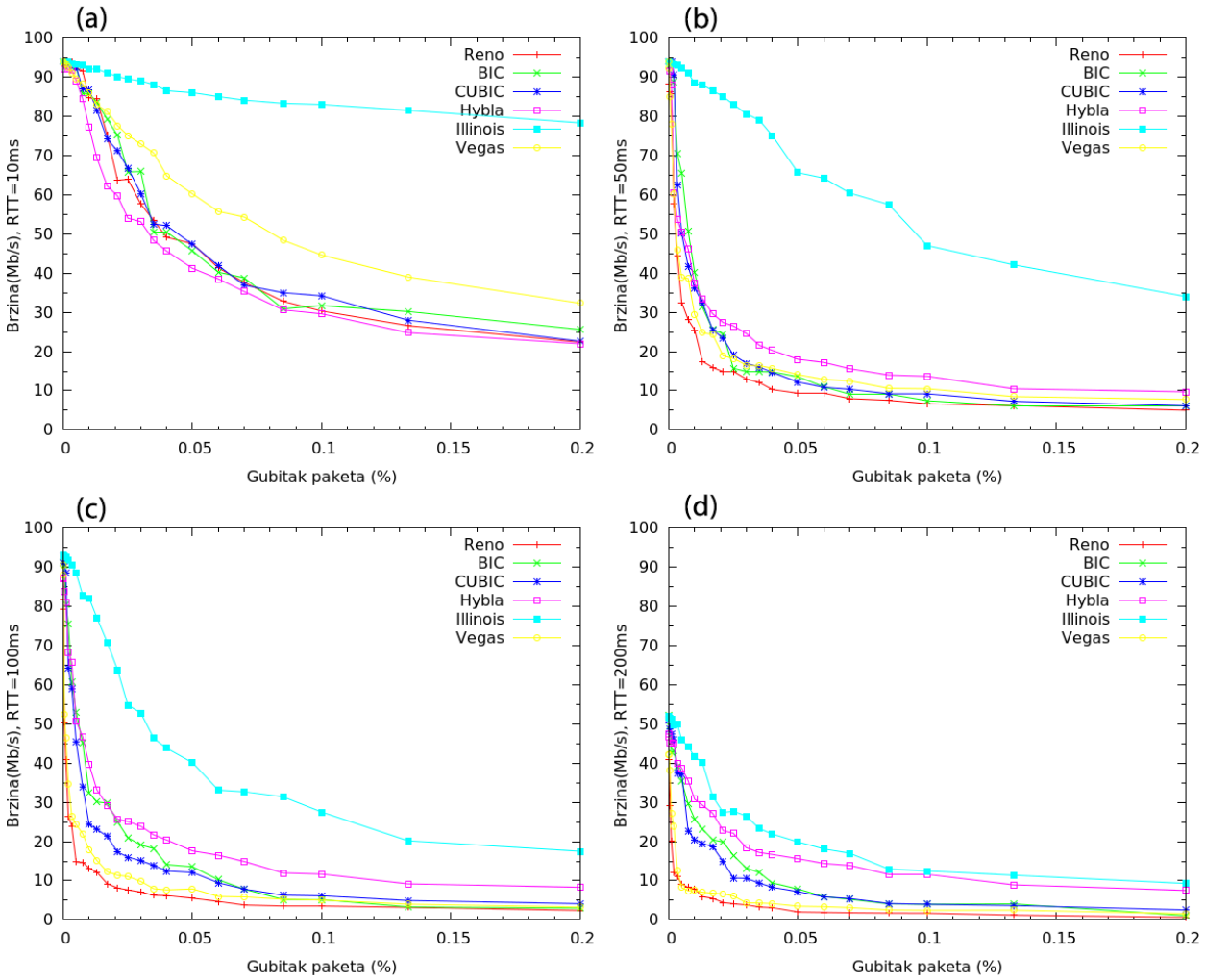
Četiri testa su urađenja, sa RTT vrednostima od 10ms, 50ms, 100ms i 200ms. Gubitak paketa je u sitnim intervalima obuhvatao vrednosti od 0% do 0,2%, za sva četiri testa. Test se sačinjavao od 150 sekundi slanja podataka sa jednog računara na drugi i merenja prosečne brzine protoka. Za merenje je korišćen program iperf [38]. Za svaki algoritam i svaku kombinaciju RTT-a i vrednosti za gubitak paketa, test je ponovljen 100 puta, sa 10 sekundi pauzom između. Na kraju, prosečna vrednost je uzeta kao reprezentativna.

Druga grupa testova je urađena na javnoj mreži, Internetu. Jedan korišćen računar je u Beogradu, istih specifikacija kao u prvoj grupi testova, a drugi je CLOUD server u Oregonu, SAD. Drugi računar je Intel Xeon procesor na 2.0Ghz i 4GB RAM-a, sa Linux Red Hat 6.3 operativnim sistemom. Veza između je 100Mb/s, sa 185ms prosečnim vremenom obilaska. Kako je razdaljina između dva računara zaista velika, prekookeanska, i vreme obilaska i kapacitet veze takođe veliki, test je bio odličan za utvrđivanje efikasnosti svakog algoritma u rešavanju problema velikog proizvoda kapaciteta i vremena obilaska.

Test je, takođe, trajao 150 sekundi, tokom kojih se merila brzina protoka. Za svaki algoritam, ponovljen je 500 puta, kako bi se dobile verodostojnije distribucije.

6.2 REZULTATI

Na slici 33 predstavljena je brzina prenosa podataka (y osa, Mb/s) u zavisnosti od procenta gubitka paketa (x osa) za svaki algoritam u testu sa direktnom vezom u lokalnoj mreži, gde je kašnjenje konstantno.

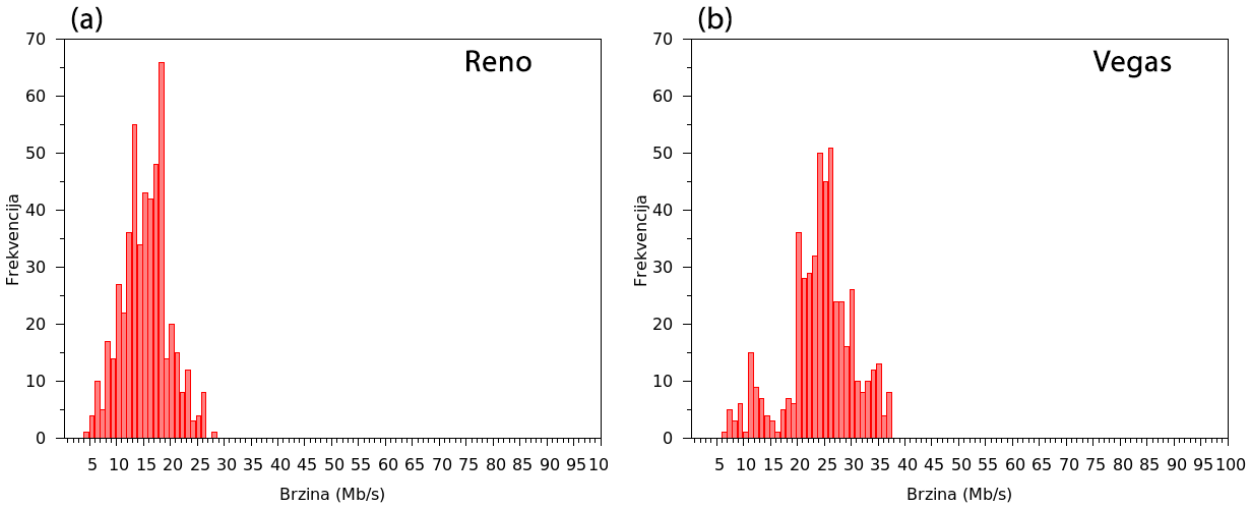


Slika 33: Rezultati testova u lokalnoj mreži, brzina prenosa podataka algoritama Reno, BIC, CUBIC, Hybla, Illinois i Vegas u Mb/s (y osa) u zavisnosti od gubitka paketa (x osa) koje je u intervalu 0% - 0,2%, sa konstantnim vremenom obilaska – (a) 10ms, (b) 50ms, (c) 100ms i (d) 200ms

U razmatranju rezultata ovakvog sintetičkog testa, treba imati na umu da nije pokušana nikakva emulacija prave mreže (druga grupa testova je rađena na pravoj mreži, pa emulacija nije bila potrebna), već je cilj bio videti kako se u različitim uslovima algoritmi ponašaju.

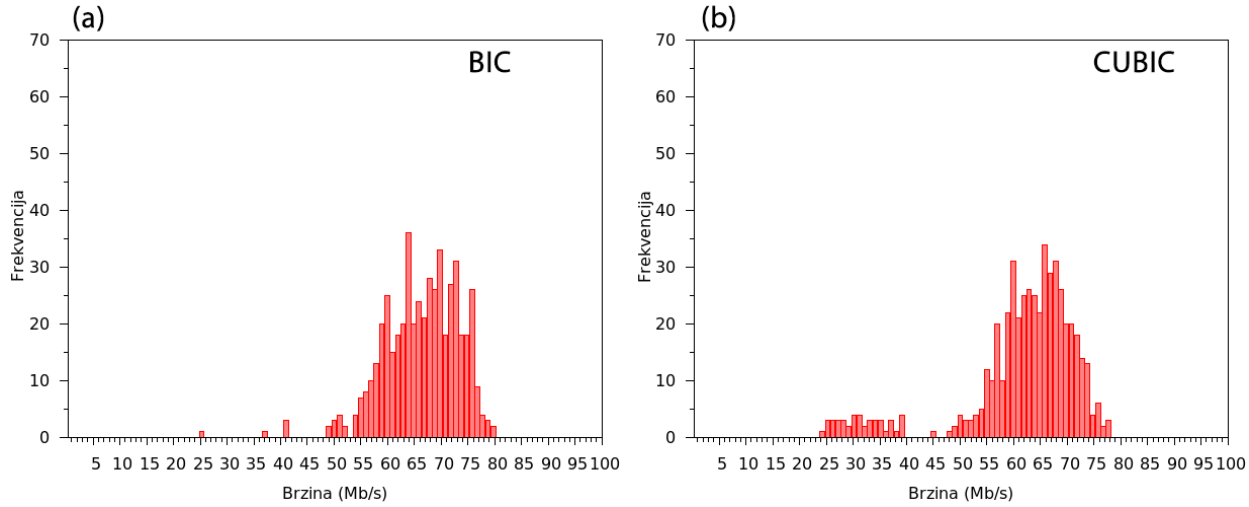
Reno je prevaziđen algoritam. To je jedno od najranijih rešenja koje nije dizajnirano za veze sa velikim propusnim opsegom i vremenom obilaska. Zato i ne čudi da je na ovom testu ostvario najlošije rezultate. BIC i CUBIC su pružili skoro iste performanse, što je i očekivano, s obzirom da im je mehanizam za rast prozora sličan. Illinois je ostvario ubedljivo najbolje rezultate. Fiksno vreme obilaksa dozvolilo je ovom algoritmu da aditivno uvećanje prozora bude veliko, a multiplikativno smanjenje malo, što se odrazilo na količinu prenetih podataka za dato vreme. Iznenađujuće je da se Vegas pokazao jako loše, iako se očekivalo da iskoristi fiksni RTT slično kao Illinois. Njegovi rezultati jedva da su bolji od Reno-a.

Na slikama 34, 35 i 36 predstavljene su distribucije brzina (x osa je brzina u Mb/s, a y osa predstavlja frekvenciju, broj pojavljivanja) pri testu na javnoj mreži i vezi Beograd – Oregon.



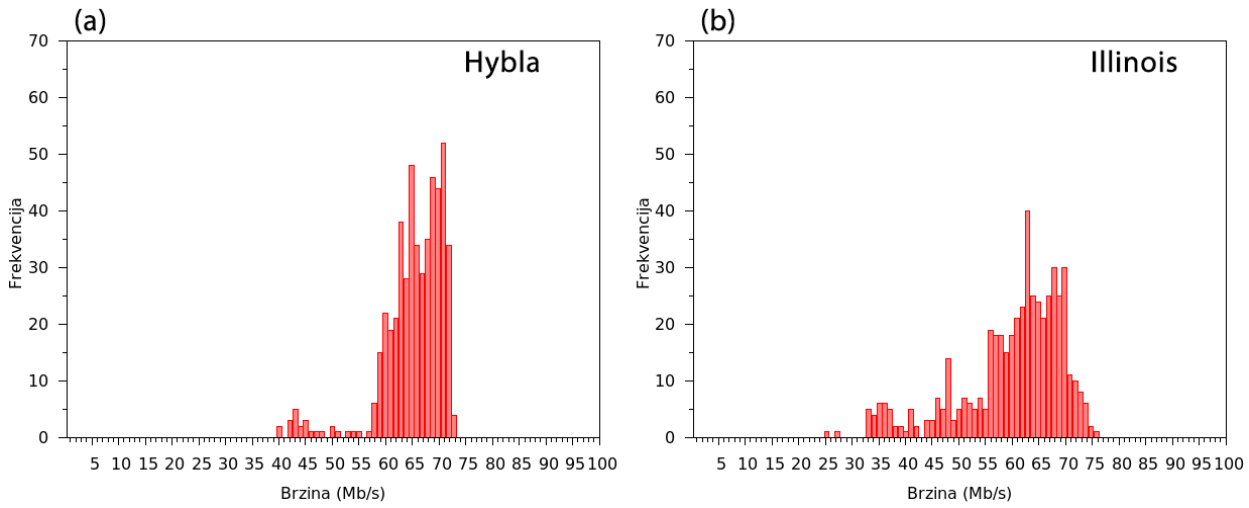
Slika 34: Rezultati testova na javnoj mreži; distribucija brzine prenosa podataka algorimana (a) Reno i (b) Vegas

Što se tiče Reno algoritma, testovi pri uslovima javne mreže na vezi sa velikim BDP-om potvrđuju da je gotovo neupotrebljiv. Prosečna brzina je 15,8Mb/s. Vegas jeste pružio nešto bolje rezultate, ali i dalje nedovoljno dobre, što pokazuje da je skepticizam ka proaktivnim algoritmima opravdan. Prosek je 24,45 Mb/s.



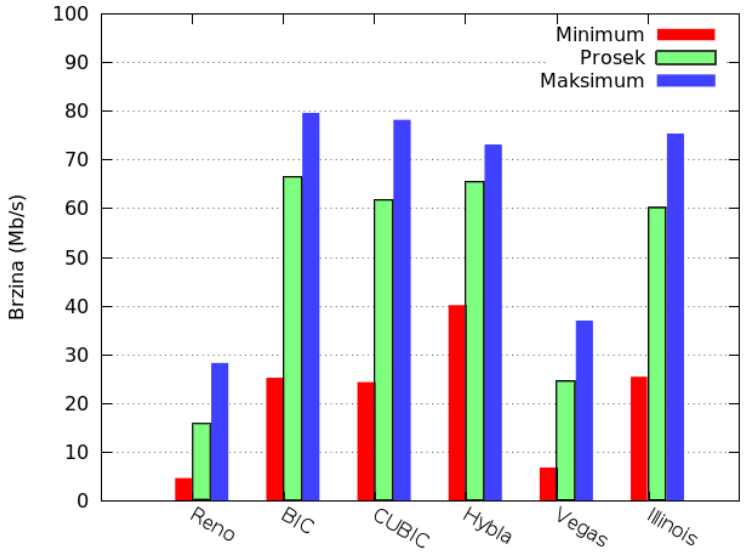
Slika 35: Rezultati testova na javnoj mreži; distribucija brzine prenosa podataka algorimana (a) BIC i (b) CUBIC

BIC i CUBIC su ostvarili prilično dobre rezultate. Prosečna brzina BIC algorima je 66,31Mb/s, što je brže od svih ostalih. CUBIC je nešto lošiji, sa prosekom od 61,6Mb/s.



Slika 36: Rezultati testova na javnoj mreži; distribucija brzine prenosa podataka algorimana (a) Hybla i (b) Illinois

Hybla je ostvarila gotovo iste prosečne rezultate kao i BIC, 65,34Mb/s, s tim da je disperzija u distribuciji brzina bila znatno manja, što dokazuje veću stabilnost algoritma. Illinois ima prosek od 60,08Mb/s, što je nešto niže od BIC-a, CUBIC-a i Hybla-e, ali je prilično dobro, ako uzmemo u obzir da ovaj algoritam uopšte ne rešava problem nefer alokacije prema RTT-u, pa ta činjenica ostavlja mesta za dalju optimizaciju.



Slika 37: Rezultati testova na javnoj mreži; minimalna, prosečna i maksimalna brzina prenosa svakog algoritma

7 ZAKLJUČAK

Ovim radom je pokušano da se na pregledan i celovit način opiše problem zagušenja i njegovo rešavanje.

Prvo je ilustovana struktura TCP protokola, bez ulaženja u implementacione detalje, kako bi se razumeli osnovni principi pouzdanog prenosa podataka. Definisan je pojam zagušenja i opisane situacije kada se ono javlja i dati su uslovi za „korektno ponašanje“ na mreži, u smislu ne dozvoljavanja da se ista preoptereti preko maksimalne granice.

Razmatrana su klasična rešenja, sa kojima problem zagušenja u užem smislu nestaje, mreža se ne opterećuje više nego što može da podrži. Objašnjeni su glavni principi i mehanizmi koji se koriste i danas. Nakon toga, pokazano je da isti algoritmi imaju problem da postignu efikasan prenos i zašto. Dva glavna problema su opisana, veliki proizvod kapaciteta i vremena obilaska, i nefer alokacija prema vremenu obilaska.

Zagušenje u širem smislu je težak problem za rešavanje. Treba omogućiti da se brzo i u potpunosti iskoriti dostupan kapacitet, a istovremeno ne preoptereti mreža previše agresivnim pristupom. Razlog velike težine problema je nedostatak jasne signalizacije o stanju na mreži, a i činjenica da bilo kakav signal kasni određeno vreme. U cilju rešavanja, predloženo je pet modernih algoritama. Kroz njih su opisani različiti pristupi i metode. Iz rezultata testova, ne može se reći koji je algoritam najbolji, s obzirom na to da su ostvarili slične rezultate. Najvažnije, BIC, CUBIC, Hybla i Illinois su pokazali ogroman napredak u postizanju veće efikasnosti u odnosu na Reno algoritam i svakako se ne bi pogrešilo ako bi se on zamenio bilo kojim od njih.

Dalja unapređenja su koncentrisana na Illinois algoritmu, pošto je on bio najinteresantniji za proučavanje, davao prilično dobre rezultate sa svojom „osnovnom” verzijom, a primećeno je da mesto za poboljšanje postoji. Primećeno je da α_{min}, α_{max}, β_{min} i β_{max} konstante mogu biti pogodnije nameštene kako bi se kapacitet bolje iskoristio. Trenutni rad se sastoji iz optimizovanja tih parametara i funkcija koje određuju α i β, kako bi se Illinois napravio agresivnijim kada je „put slobodan”, pri tom zadržavajući fleksibilnost i umerenost na pojavu zagušenja. Radi se i na eliminaciji zavisnosti veličine prozora od vremena obilaska, na način sličan kao kod Hybla-e. Preliminarni rezultati pokazuju blizu 30% bolje performanse u odnosu na originalnu verziju Illinois algoritma, što podstiče autora ovog teksta na dalji rad.

8 REFERENCE

[1] TRANSMISSION CONTROL PROTOCOL, RFC 793, Sep. 1981.

[2] K. Thompson, G. Miller, R. Wilder, „Wide-area internet traffic patterns and characteristics“, IEEE, vol. 11, no. 6, pp. 10–23, Nov. 1997.

[3] K. Claffy, G. Miller, K. Thompson, „The nature of the beast: Recent traffic measurements from an internet backbone“, INET'98, Network Technology and Engineering, Jul 1998.

[4] S. Saroiu, P. K. Gummadi, S. D. Gribble, „A Measurement study of peer-to-peer le sharing systems“, Proceedings of Multimedia Computing and Networking 2002, vol. 4673, pp. 156–170, Jan. 2002.

[5] K. Sripanidkulchai, B. Maggs, H. Zhang, „An analysis of live streaming workloads on the Internet“, Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, New York, NY, USA, ACM Press, pp. 41–54, 2004.

[6] S. Liu, T. Basar, R. Srikant, „TCP-Illinois - A Loss and DelayBased Congestion Control Algorithm for High-Speed Networks“, Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools, 2006.

[7] D. X. Wei, P. Cao, S. H. Low, „Time for a TCP Benchmark Suite?“, Technical report, Caltech, Avg. 2005.

[8] C. Barakat, E. Altman, and W. Dabbous, „On TCP Performance in a Heterogenous Network: A Survey“, IEEE Communications Magazine, Jan. 2000.

[9] S. Floyd, „Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic“, Computer Communication Review, Okt. 1991.

[10] T.V. Lakshman and U. Madhow, „The performance of TCP/IP for networks with high bandwidth-delay products and random loss“, IEEE/ACM Transactions on Networking, Jun 1997.

[11] E. Altman, C. Barakat, E. Laborde, P. Brown, D. Collange, „Fairness Analysis of TCP/IP“, Proceedings of IEEE Conference on Decision and Control, vol. 1, pp. 61-66, Dec. 2000.

[12] V.. Cerf, and R. Kahn, „A Protocol for Packet Network Intercommunication“, Communications, IEEE Transactions on, Volume:22 , Issue: 5, pp. 637 – 648, Maj 1974.

[13] Architectural Principles of the Internet, RFC 1958, Jun 1996.

[14] S. Floyd, K. Fall, „Promoting the Use of End-to-End Congestion Control in Internet“, IEEE/ACM Transactions on Networking, Avg. 1999.

[15] V. Jacobson, „Congestion avoidance and control“, ACM SIGCOMM, pp. 314-329, Avg. 1988.

[16] Open Research Issues in Internet Congestion Control, RFC 6077, Feb. 2011.

[17] S. Floyd, „Connections with Multiple Congested Gateways in PacketSwitched Networks Part 1: One-way Traffic“, Computer Communication Review, Okt. 1991.

[18] TCP Congestion Control, RFC 5681, Sep. 2009.

[19] The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 3782, Apr. 2004.

[20] K. Tan, J. Song, Q. Zhang, M. Sridharan, „A Compound TCP Approach for High-speed and Long Distance Networks“, IEEE INFOCOM, Apr .2006.

[21] E. de Souza, D. Agarwal, „A HighSpeed TCP Study: Characteristics and Deployment Issues“, LBNL Technical Report 53215

[22] S. Floyd, „Highspeed TCP for large congestion window“, Internet Draft draft-floyd-tcp-highspeed-01.txt, work in progress, Feb. 2003.

[23] D. X. Wei, P. Cao, S. H. Low, „Time for a TCP Benchmark Suite?“, Technical report, Caltech, Avg. 2005.

[24] TCP Selective Acknowledgment Options, RFC 2018, Okt. 1996.

[25] M. Mathis, J. Mahdavi, „Forward Acknowledgment: Refining TCP Congestion Control“, ACM SIGCOMM Computer Communication Review Homepage, vol. 26, issue 4, pp. 281-291, Okt. 1996.

[26] L. Xu, K. Harfoush, and I. Rhee, „Binary increase congestion control (BIC) for fast long-distance networks“, in IEEE INFOCOM 2004, 2004.

[27] I. Rhee, L. Xu, „CUBIC: A New TCP-Friendly High-Speed TCP Variant“, ACM SIGOPS Operating Systems Review, Vol. 42, Issue 5, pp. 64-74, Jul 2008.

[28] C. Caini and R. Firrincieli, „TCP Hybla: a TCP enhancement for heterogeneous networks“, International Journal of Satellite Communication and Networking, vol. 22, no. 5, pp. 547–566, Sep. 2004.

[29] L. S. Brakmo, S. W. O'Malley, L. L. Peterson, „TCP Vegas: New Techniques for Congestion Detection and Avoidance“, SIGCOMM Comput. Commun. Rev., Vol. 24, No. 4., pp. 24-35, Okt. 1994.

[30] L. S. Brakmo, S. W. O'Malley, L. L. Peterson, „TCP Vegas: New Techniques for Congestion Detection and Avoidance“, SIGCOMM Comput. Commun. Rev., Vol. 24, No. 4., pp. 24-35, Okt. 1994.

[31] J. Martin, A Nilsson, I Rhee, „Delay-Based Congestion Avoidance for TCP“, IEEE/ACM Trans. Netw., Vol. 11, No. 3., pp. 356-369, Jun 2003.

[32] S. Liu, T. Basar, R. Srikant, „TCP-Illinois - A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks“, Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools, 2006.

[33] K. Yamada, R. Wang, M.Y. Sanadidi, and M. Gerla, „TCP Westwood with Agile Probing: Dealing with Dynamic, Large, Leaky Pipes“, IEEE Journal on Selected Areas in Communications 23(2): 235-248, 2005.

[34] A. Kuzmanović, and E. W. Knightly, „TCP-LP: Low-Priority Service via End-Point Congestion Control”, Proceedings of IEEE INFOCOM 2003, San Francisco, CA, Apr. 2003.

[35] A. Venkataramani, R. Kokku, and M. Dahlin, „TCP Nice: A Mechanism for Background Transfers”, 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, USA, Dec. 2002.

[36] F. Wang and Y. Zhang, „Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response”, ACM MobiHoc'02, pp. 217-225, Lausanne, Switzerland, Jun 2002.

[37] iperf http://iperf.sourceforge.net

[38] tc http://linux.die.net/man/8/tc