UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andrej Mohar

Nadzor zamašitev v hitrih omrežjih

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr. Mojca Ciglarič

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: 00193/2012
Datum: 02.02.2012

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

Kandidat: **ANDREJ MOHAR**

Naslov: **NADZOR ZAMAŠITEV V HITRIH OMREŽJIH**
CONGESTION CONTROL IN LONG FAT NETWORKS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Opišite nadzor zamašitev, kot ga poznamo v najpogosteje uporabljenih različicah protokola TCP. Pojasnite, zakaj so pri uporabi teh različic v hitrih omrežjih težave z neučinkovito izrabo razpoložljive pasovne širine. V literaturi poiščite predloge za izboljšan nadzor zamašitev, ki bi bil prilagojen na omrežja z dolgimi povezavami (večja latenca) in visoko pasovno širino. Izberite primeren simulator omrežnega prometa in izberite najbolj zanimive različice protokola TCP z izboljšanim nadzorom zamašitev. Klasične in izboljšane različice na preprosti topologiji preizkusite v simulatorju omrežnega prometa. Opazujte več zanimivih parametrov učinkovitosti in kritično ovrednotite rezultate simulacij.

Mentor:
doc. dr. Mojca Ciglarič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Andrej Mohar,

z vpisno številko 63020333,

sem avtor/-ica diplomskega dela z naslovom:

Nadzor zamašitev v hitrih omrežjih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom doc. dr. Mojca Ciglarič
- 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 13.08.2012 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se svoji mentorici doc. dr. Mojci Ciglarič za posvečen čas ter za strokovno pomoč in nasvete pri izdelavi naloge.

Zahvaljujem se tudi moji družini za potrpežljivost in podporo pri dokončanju študija.

Staršem

Kazalo

Seznam uporabljenih kratic in simbolov

ARPA - Advanced Research Projects Agency (agencija, katera je razvila ARPANET)
TCP - Transmission Control Protocol
IP - Internet Protocol
MSS - Maximum segment size (določa največjo velikost segmenta, katerega lahko računalnik ali omrežni vmesnik obdela)
RTO - Round Trip Timeout (čas, določen za čakanje preden se paket ponovno pošlje)
RTT - Round Trip Time (čas od trenutka pošiljanja paketa do trenutka sprejema potrditve tega paketa)
RTTM - Round Trip Time Measurement (Algoritem za merjenje RTT časa)
cwnd - Congestion Window (Okno za pošiljanje, TCP ga uporablja pri preprečevanju zamašitev)
ssthresh - Slow Start Threshold (prag, pri katerem se algoritem počasnega začetka konča in začne algoritem izogibanja zamašitvam)
BDP - Bandwidth - Delay Product (produkt pasovne širine in latence, merjene v eni smeri med odjemalcem in strežnikom)
LFN - Long Fat Networks (omrežja z velikim BDP produktom)
IEEE - Institute of Electrical and Electronics Engineers
AIMD - Additive Increase Multiplicative Decrease - način povečevanja in zmanjšanja okna za pošiljanje
SACK - Selective Acknowledgements - način potrjevanja prispelih paketov

Povzetek

V današnjih časih je uporaba TCP protokola za prenos podatkov zelo razširjena. Ljudje nenehno poskušajo pohitriti prenos podatkov po Internetu. Hitrost omrežjih ni odvisna samo od hitrih povezav, temveč tudi o kvaliteti uporabe teh povezav.

Leta 1969 se je zgodil prvi veliki zamašitveni kolaps na ARPANET omrežju. Tedaj je osnovni različici TCP protokola dodan algoritem, ki upočasnjuje pošiljanje podatkov in tako preprečuje zamašitve v omrežjih. Od tedaj naprej se izdelujejo TCP algoritmi, ki poskušajo čim boljše določiti pravilno hitrost pošiljanja podatkov tako da bo omrežje popolnoma uporabljeno, brez da bi ogrožali zamašitve v omrežjih.

Diploma opisuje osnovne elemente TCP protokola, ki sodelujejo v preprečevanju zamašitev v omrežjih. Predstavljene so nove hitre vrste omrežjih ter problemi, ki nastajajo pri komunikaciji pri takšnih omrežjih. Na koncu so našteti in na kratko opisani simulatorji in simulacije, izvedene na znanim različicama TCP algoritma.

Rezultati simulacij prikazujejo različne lastnosti algoritmov, in sicer spremembe okna za pošiljanje, uporabo vrste, zakasnitve v vrsti, izpade paketov iz vrste, povratni čas paketov in uporabljeno pasovno širino. Simulacije so izvedene na enostavnem omrežju z ozkim grlom. Simulirani so naslednji protokoli: TCP Reno, TCP Vegas, TCP Cubic, TCP Hybla in TCP Compound. Najmanjše število poslanih in sprejetih paketov ima TCP Reno. TCP Vegas ima najboljše rezultate, ampak v simulaciji niso prisotne ostale vrste podatkovnih tokov. Naslednji po vrsti je TCP Cubic, ki je primeren za fiksne kot tudi za brezžične povezave.

Ključne besede:

TCP, zamašitev, omrežje, povezava, podatkovni tok, simulacija, simulator

Abstract

In the present time the TCP protocol is used widely for the data transmission. People are constantly trying to speed up the data transmission over the Internet. The networking speed is not dependent on the fast Internet networks alone, but on the connection quality as well.

The first big congestion collapse happened in the year 1969 on the ARPANET network. After that event the new algorithm was added to the base TCP protocol, which slows down the sending rate thus stops congestion from happening. From that point on, the new TCP variants are being developed, which try to predict the right sending speed so that the network will be fully utilized without major congestion collapses.

The thesis describes the basic TCP protocol elements, which participate in the congestion avoidance process. It introduces the modern fast networks and the problems which arise during the communication on such fast networks. The different simulators and simulations are enumerated and described in the last part of the thesis.

The simulation results describe the similarities and differences on several different graphs: congestion window, the queue usage, the queue delays, dropped packets, round trip times and bandwidth usage. The simulations are used on a simple model containing the "chokepoint" router. The following protocols are simulated: TCP Reno, TCP Vegas, TCP Cubic, TCP Hybla and TCP Compound. The TCP Reno has the lowest number of sent and received packets. The results show the TCP Vegas as the best algorithm, but no other flows are present in the simulation. The second best is the TCP Cubic protocol, which can be used on both wired and wireless networks.

Key words:

TCP, congestion, network, connection, data flow, simulation, simulator

Poglavje 1

Uvod

Prvo omrežje s paketno komunikacijo je nastalo leta 1969 v ZDA zaradi omogočanja dostopa do tedaj majhnega števila močnejših raziskovalnih računalnikov. Poimenovali so ga *ARPANet*, po agenciji *Advanced Research Projects Agency* (ARPA). Tekom naslednjih let se je število računalnikov priklopljenih na omrežje hitro povečevalo. V oktobru leta 1986 se je zgodil prvi zastoj paketnega prometa v omrežju [1]. V času zastoja, pretoka podatkov čez omrežje skoraj da ni bilo. Na prehode je v določenem trenutku prihajalo več paketov kot jih je omrežje lahko obvladalo. Od tedaj naprej se razvijajo algoritmi in protokoli, ki poskušajo omejiti hitrost pošiljanja na takšen način, da bo čim manj izpadov paketov in da bo hkrati čim večja uporaba pasovne širine omrežja.

Zaradi čim boljše in hitrejše povezave se tehnologije Interneta spreminjajo in izboljšujejo vsak dan. Od velikega pomena je izboljšanje hitrosti in pretoka podatkov pri točkah najslabše prehodnosti, kot so usmerjevalniki in prehodi. Namen tega diplomskega dela je predstaviti postopke in algoritme za kontrolo in preprečevanje zamašitev na teh točkah. Opisana so tudi sodobna hitra omrežja, kot tudi problemi, ki nastajajo pri komunikaciji *Transmission Control Protocol* (TCP) protokolov na takih omrežjih in njihove rešitve. Glavni cilj diplomskega dela je predstavitev simulacij starejših in sodobnih različic TCP protokola na enostavnem simulacijskem modelu, z idejo poskusa simulacij tudi na najnovejših hitrih 70 Gb/s in 100 Gb/s omrežjih.

Diplomsko delo je sestavljeno iz uvoda in petih poglavjih. Drugo poglavje opisuje osnovne dele TCP protokola, ki sodelujejo pri reševanju problema zamašitev v omrežjih. Opisano je TCP zaglavje in postopek komunikacije. Na koncu poglavja so opisane zamašitve omrežja. Tretje poglavje predstavlja nove vrste zelo hitrih omrežjih in probleme, ki lahko nastanejo pri komunikaciji po teh omrežjih ter njihove rešitve. V četrtem poglavju so našteti in na kratko

opisani sodobni simulatorji za simuliranje omrežjih. Opisane so tudi teorijske osnove za izdelavo simulacij. Peto poglavje opisuje delovanje pet različic TCP algoritma ter predstavlja izvedene simulacije in njihove rezultate. Šesto poglavje na splošno opisuje rezultate in opažanja ter predlaga smeri nadaljevanja raziskav in simulacij.

Poglavje 2

Osnove delovanja TCP protokola in problem zamašitev

2.1 Osnove TCP/IP povezav in TCP protokola

TCP je narejen kot dodatek na *Internet Protocol* (IP) [2, 3, 4]. Glavna naloga IP protokola je dostaviti paket od pošiljatelja do prejemnika, TCP pa skrbi za zanesljivost pri prenosu podatkov. Je povezavno usmerjen protokol. To pomeni da so za uspešni prenos podatkov nujna tri koraka: vzpostavljanje povezave, prenos paketov in zaključek povezave. Zgradba TCP protokola omogoča, da se prilega v plastno hierarhijo komunikacijskih protokolov. Pri TCP/IP komunikaciji se uporabljajo tako imenovane vtičnice. Vtičnica je sestavljena od enega IP naslova in enih TCP vrat. Po uspešni povezavi, dve vtičnici lahko med sabo komunicirata.

TCP pošilja podatke v obliki nizov skupin od osem bitov (v nadaljevanju bo skupina osem bitov poimenovana oktet. Pri večini današnjih računalnikov je to enako enem bajtu). Vsako sporočilo za pošiljanje se razdeli na manjše nize oktetov, ki se imenujejo segmenti. TCP segment se potem lahko razdeli več IP paketov, ki ohranijo TCP zaglavje in dodajo še IP zaglavje. Takšni paketi so pripravljeni za pošiljanje.

V nadaljevanju bodo na kratko opisani deli TCP protokola, ki sodelujejo pri nastanku ali reševanju problema zamašitev. IP protokol ne bo opisan ker ne sodeluje v nadzoru in preprečevanju zamašitev. Za vse izpuščene detajle kot tudi za detajle IP protokola se bralec lahko posvetuje z navedeno literaturo.

2.1.1 TCP zaglavje

Slika 2.1: Podatki v zaglavju TCP paketa omogočajo njegovo zanesljivo pošiljanje in sprejem. Številke v glavi tabele označujejo vrstni red posameznega bita.

Vsak TCP paket ima svoje zaglavje, ki vsebuje informacije pomembne za zanesljivo dostavljanje paketa. Zaglavje TCP protokola je prikazano v tabeli 2.1.

Kot je že omenjeno, pošiljateljeva in prejemnikova vrata (angl. *port*) so celi šestnajst-bitni števili, ki določata skupaj z IP naslovi par vtičnic, ki med sabo komunicirata. Pri komunikaciji v paketih pogosto zamenjata mesto, odvisno od smeri, v katero paket potuje. Prva številka je zmeraj številka pošiljateljevih vrat, druga je številka prejemnikovih vrat.

Številka vrstnega reda ali številka sekvence (angl. *sequence number*) je dvaintrideset-bitno vrstno število prvega okteta v trenutnem segmentu. S pomočjo tega števila TCP lahko zbere vse prispele pakete in jih po vrsti sestavi v začetno sporočilo. Poleg tega se to število uporabi za določanje manjkajočih ali duplih paketov.

Številka potrditve (angl. *acknowledgement number*) je tudi dvaintrideset-bitno število, ki se uporablja za potrjevanje prispetja paketa. Predstavlja število naslednjega pričakovanega okteta. Na ta način je pravzaprav potrjeno prispetje vseh segmentov do tega vrstnega števila. Če kateri paket zamuja ali se izgubi, sledeči paketi pa uspešno prispejo, TCP potrjuje samo del sporočila, ki je kompletno in točno sprejet. Ostali paketi se hranijo dokler ne pride pričakovani paket.

Ker sta obe številki dvaintrideset-bitni, pomembno je poudariti, da pri matematičnih operacijah lahko pride do preliva (angl. *overflow*). Zaradi tega se vse matematične operacije na teh številih izvajajo po modulu 2^{32}.

Odmik (angl. *data offset*) je štiri-bitno število vrstic v TCP zaglavju, ki pove kdaj se konča zaglavje in kdaj se začnejo podatki. Vsaka vrstica ima dvaintrideset bitov oziroma štiri okteta. Vsako TCP zaglavje ima vsaj pet obveznih vrstic, torej najmanjši možni odmik je pet. Na to se še prištejejo vrstice za morebitne možnosti. Največje štiri-bitno število je 15, to je tudi največje število vrstic v TCP zaglavju. Možnosti torej lahko zasedejo največ deset vrstic ali štirideset bajtov.

V originalni različici TCP protokola za odmikom sledi šest bitov prostora, rezerviranega za bodočo uporabo.

Potem sledi šest zastavic. Zastavice so eno-bitne vrednosti, ki označujejo nekatero od spodaj navedenih akcij:

- URG (angl. *urgent*) zastavica se uporablja za označevanje segmentov, ki vsebujejo prioritetne podatke. To pomeni, da program, ki lahko sprejema take podatke, mora biti sposoben dati prednost obdelavi nujnih podatkov. Postavitev URG zastavice še ne pomeni, da bo segment takoj poslan. Zaradi tega se pogosto uporablja skupaj s PSH zastavico.

- ACK (angl. *acknowledgement*) zastavica je danes postavljena v skoraj vsakem TCP paketu. Služi skupaj s številko potrditve za potrjevanje vseh uspešno sprejetih podatkov do trenutka pošiljanja tega paketa.

- PSH (angl. *push*) zastavica se je v preteklosti uporabljala za takojšnje pošiljanje majhnih segmentov. Včasih je namreč TCP protokol čakal, da se zbere čim več podatkov, tako da se okno dovoljeno za pošiljanje popolni. Pošiljanje večjih segmentov je pomenilo hitrejšo in bolj optimalno povezavo. Aplikacija je lahko sama določila kdaj bo ta zastavica postavljena. V današnjih časih to ni več možno, TCP protokol pa sam določa kdaj bo postavljena. Ponavadi je ta zastavica postavljena na vsakem paketu, ker zapisovanje paketa danes pomeni tudi takojšnje pošiljanje paketa.

- RST (angl. *reset*) zastavica se pošilja v primeru napake pri povezovanju ali v primeru, da ena stran v povezavi ima nepričakovane probleme z izvajanjem.

- SYN (angl. *synchronize*) zastavico pošlje vsaka vtičnica samo enkrat na začetku povezave. Na ta način pove, da je pripravljena za komunikacijo.

- FIN (angl. *finish*) zastavica se uporablja pri uspešnem zaključku povezave.

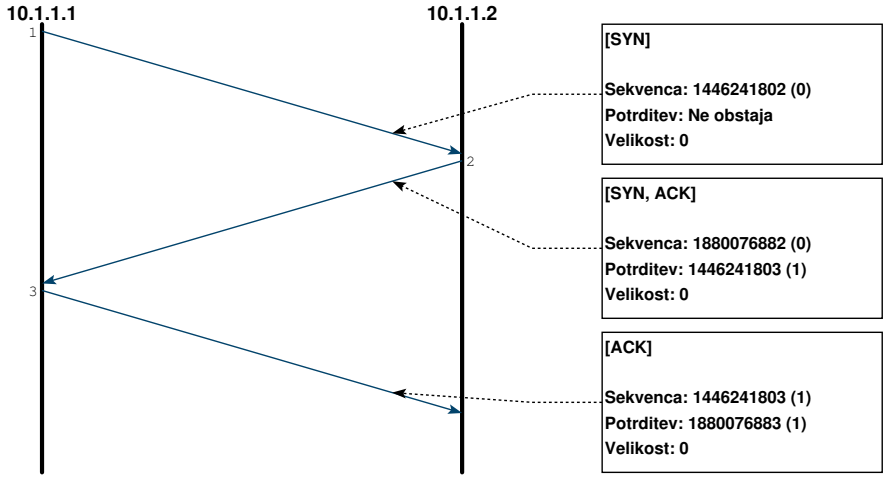
Okno (angl. *window*) je šestnajst-bitna številka, ki se uporablja pri kontroli pretoka (angl. *flow control*) podatkov. V primeru, da pošiljatelj pošilja hitrejše pakete kot jih prejemnik lahko obdeluje, prejemnik lahko kontrolira hitrost pošiljanja s pomočjo okna. Pošiljatelju okno pove, koliko oktetov je lahko prejemnik še obvladal v trenutku pošiljanja tega paketa. Pošiljatelj lahko pošlje samo toliko oktetov, koliko mu prejemnik dovoli. Če je okno enako nuli, pošiljatelj ne sme pošiljati podatkov do spremembe velikosti okna.

Kontrolna vsota (angl. *checksum*) je šestnajst-bitno število za kontrolo točnosti podatkov. Izračuna se pred pošiljanjem paketa in se zapiše v zaglavje. Pri sprejemu paketa se izračuna še enkrat in se primerja s seštevkom iz zaglavja. Če sta enaki, je velika verjetnost, da se nobeden del podatkov v paketu ni slučajno spremenil ali uničil pri pošiljanju.

Kazalec "urgent" deluje skupaj s prej opisano URG zastavico pri prenosu prioritetnih podatkov. Prioritetni podatki se začnejo s številko vrstnega reda. Kazalec "urgent" pove koliko je od začetka segmenta do zadnjega okteta prioritetnih podatkov. Torej, če se seštejejo številka vrstnega reda in kazalec "urgent", se dobi položaj zadnjega prednostnega znaka v trenutnem segmentu. Zatem lahko še sledijo morebitni navadni podatki, ki niso označeni kot prednostni.

Vsi do sedaj navedeni deli so definirani v začetni TCP specifikaciji [3] kot nujni deli vsakega TCP zaglavja. Na koncu zaglavja lahko še sledijo morebitne možnosti, ki si jih med sabo izmenjujeta dve vtičnici, za hitrejšo ali bolj zanesljivo delovanje TCP protokola. Ker se je sčasoma TCP protokol izboljševal in spreminjal, so različnim variantam TCP protokola dodajali nove možnosti. Zaradi tega je možno, da nekatere različice TCP protokola ne razumejo vseh prispelih možnosti. Pri komunikaciji se ponavadi uporabljajo samo tiste možnosti, ki jih obe strani lahko uporabljata. Kot je že prej omenjeno, možnosti lahko maksimalno zasedejo deset vrstic ali štirideset oktetov v TCP zaglavju. Če velikost možnosti ni deljiva z velikostjo vrstice (4 okteta), za možnostima morajo slediti ničle do konca vrstice.

Najpogostejše uporabljana možnost je maksimalna velikost segmenta (angl. *maximum segment size* ali skrajšano MSS). Ta možnost določa največjo velikost enega segmenta brez TCP in IP zaglavja. Nastavi se lahko samo v SYN paketu. Če ta možnost ni nastavljena, vsak TCP protokol jo samodejno nastavi na nekatero vnaprej določeno vrednost, najpogostejše 536 bajtov.



Slika 2.2: Povezava med TCP vtičnicami se začne s pošiljanjem treh paketov. Sekvence in potrditve so prikazane absolutno in relativno (v oklepajih). Potrjevanje ACK paketa povzroča povečanje zaporedne številke potrditve za ena.

2.1.2 Vzpostavljanje TCP/IP povezave

Sam TCP/IP protokol ne določa način na kateri bo uporabljen znotraj različnih aplikacij. Določa edino kako bo potekala komunikacija in pretok podatkov med aplikacijami, ki med seboj komunicirajo. Najpogostejša oblika komunikacije, ki jo uporablja večina aplikacij, je uporaba modela odjemalec-strežnik (angl. *client-server*). Aplikacija, ki deluje kot strežnik, neomejeno čaka na prihajajoče povezave in streže vsako povpraševanje po povezavi s strani uporabnikov. Uporabniške vtičnice ponavadi začenjajo povezavo [5].

Povezava se vzpostavlja s pomočjo trojnega rokovanja (angl. *three-way handshake*). Uporabnikov TCP protokol (v nadaljevanju uporabnik) prvi pošlje paket s postavljeno SYN zastavico. V tem prvem paketu je sekvenca posebno izračunano število z naključno komponento, ki strežniku pove začetno sekvenco uporabnika [6]. Številka potrditve še ni nastavljena (oziroma je v paketu nastavljena na nulo), ker še ni sprejeta številka sekvence strežnika. Uporabnik lahko pošlje še morebitne možnosti, opisane v odstavku 2.1.1.

V drugem koraku strežnikov TCP protokol (v nadaljevanju strežnik) pošlje odgovor v obliki paketa z nastavljenimi SYN in ACK zastavicami. Na ta način pove da je pripravljen za komunikacijo. Strežnik si prav tako izbere naključno

številko za začetno sekvenco, za številko odgovora pa pošlje uporabnikovo sekvenco povečano za ena. Če strežnik uporabi dodatne možnosti, ki jih je sprejel od uporabnika, v svojem odgovoru lahko pošlje svoje nastavitve za te možnosti. Na ta način se uporabnik in strežnik zmenita za točen način komunikacije.

Kot zadnji korak pri vzpostavi povezave uporabnik pošlje paket z nastavljeno zastavico ACK, s katero potrdi sprejem sinhronizacijskega paketa od strežnika. Številka sekvence je v tem paketu enaka številki strežnikovega odgovora, torej začetna številka sekvence uporabnika zvišana za ena. Številka potrditve je pa strežnikova številka sekvence zvišana za ena. Po tem paketu je povezava uspešno vzpostavljena.

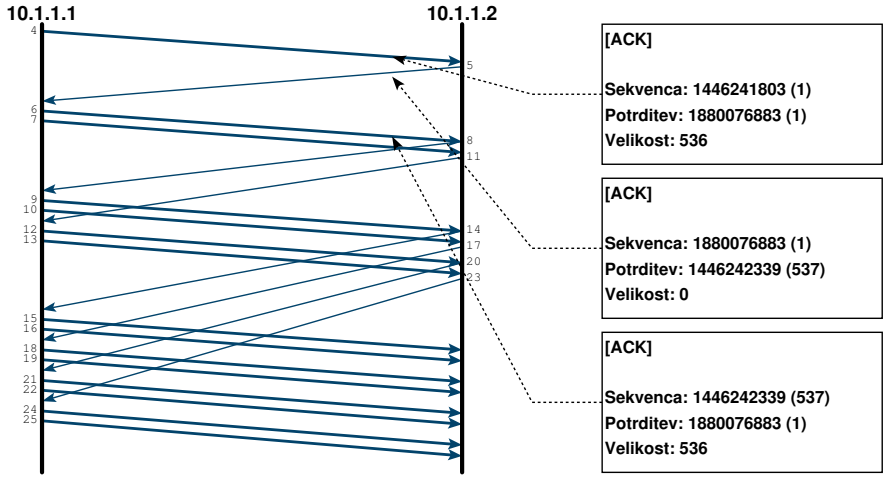
Slika 2.2 prikazuje začetek povezave med uporabnikom na naslovu 10.1.1.1 in strežnikom na naslovu 10.1.1.2.

V primeru, da vtičnica na strani strežnika ne obstaja, na začetni SYN paket TCP odgovori z ACK RST paketom. Na ta način takoj pove uporabniku da na drugi strani TCP ne sprejema odjemalce na iskani vtičnici.

2.1.3 Prenos podatkov

Po uspešno vzpostavljeni povezavi sta obe strani pripravljeni za pošiljanje paketov podatkov. Kot je že rečeno, vsaka stran ima na začetku svoje število sekvence zvišano za ena, ki je potrjeno tudi z nasprotne strani. Da bi se število sekvence zvišalo, mora biti potrjen uspešen prihod podatkov. V paketu se lahko hkrati pošiljajo podatki in potrjujejo sprejeti segmenti. Velikost samega paketa je zapisana v IP zaglavju [4]. Če paket ne vsebuje nobenih podatkov, število sekvence bo ostalo nespremenjeno za naslednji paket.

Slika 2.3 prikazuje nadaljevanje komunikacije računalnikov začete na sliki 2.2.



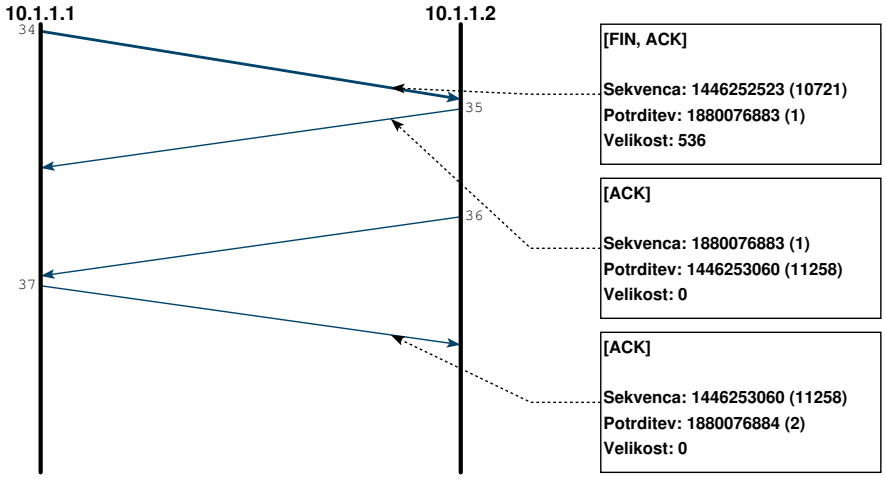
Slika 2.3: Prikaz začetnega dela prenosa podatkov. Povezava je že uspešno vzpostavljena. Debelejše puščice označujejo prehod paketov s podatki. Četrti paket prenosi prvi segment od 536 oktetov podatkov. S pomočjo petega paketa strežnik potrjuje sprejem prvih 536 oktetov z relativno potrditvijo 537. Potem se pošlje naslednjih 536 oktetov. Tako se nadaljuje dokler se ne pošljejo vsi podatki.

2.1.4 Zaključek TCP povezave

Povezava se pravilno zaključi z uporabo FIN paketa. Če sta obe strani pripravljeni za konec komunikacije, povezava se lahko konča z uporabo treh paketov. Prvi paket ima nastavljeno zastavico FIN in ga ponavadi pošlje uporabnik. Drugi paket potrdi prispevek FIN paketa s zastavico ACK in nastavi še svojo FIN zastavico. Na ta način pove, da je tudi ta stran končala pošiljanje. Na koncu še sledi potrditev ACK s nasprotne strani.

Povezava se navadno zapira z uporabo štirih paketov, kot je prikazano na sliki 2.4. Kadar prispe prvi FIN paket, TCP ga takoj potrdi in pošlje informacijo o prispetju FIN paketa aplikaciji na višjih plasteh. To pomeni da iz smeri, iz katere je prišel fin paket, podatki ne bodo več prihajali. Potem se podatki lahko pošiljajo samo v nasprotno smer. Kadar so vsi podatki poslani se kanal zapre tudi v drugo smer na isti način. Pošlje FIN paket, katerega nasprotna stran potrdi. V tem trenutku je povezava zaključena.

Potrjevanje paketa z FIN zastavico zviša število potrditve za ena, isto kot potrjevanje SYN paketa. Sprejetje SYN ali FIN zastavice zmeraj zviša število



Slika 2.4: Prikaz zaključka TCP povezave. Prikazani so samo paketi, pomembni za uspešen zaključek povezave. V prikazanem primeru paket 34, ki začenja proces zaključevanja povezave, še prenaša podatke. Številka potrditve strežnika (paket 35) je enaka seštevku sekvence in velikosti podatkov v paketu 34. Potem je še zvišana za ena zaradi potrditve sprejetja FIN zastavice. Ko obe strani pošljeta FIN zastavice in potrdita sprejem nasprotne, je povezava zaključena. Končno število potrditve iz smeri uporabnika je dva, ker strežnik ni pošiljal nobenih podatkov. Potrjeno je samo sprejetje SYN in FIN zastavic.

potrditve za ena.

2.1.5 Izgubljeni paketi in TCP časovni merilec za ponovno pošiljanje

TCP meri čas od trenutka pošiljanja vsakega posameznega paketa do prihoda njegove potrditve. Če potrditvi rabi preveč časa za prihod, TCP zaključi, da je paket uničen in ga ponovno pošlje. Za optimalno komunikacijo se čas za ponovno pošiljanje (angl. *round trip timeout* ali skrajšano RTO) mora prilagajati situaciji v omrežju. Če je v omrežju polno prometa, merilec se mora nastaviti na več časa in obratno. V primeru velikega prometa in krajšega časa za ponovno pošiljanje se lahko paket, ki zamuja, označi kot izgubljen. Zaradi tega se lahko zgodijo nepotrebna ponovna pošiljanja in nastane še večja gneča v omrežju. Za določanje pravilnega časa za ponovno pošiljanje je pomembno prej omenjeno merjenje časa od pošiljanja paketa do sprejema njegove potrditve, seveda v primeru, da potrditev pride pravočasno (angl. *round trip time measurement* ali skrajšano RTTM).

Algoritem, ki se je uporabljal v začetni različici TCP protokola je narejen s pomočjo nizkoprepustnega filtra (angl. *low-pass filter*), definiranega v enačbi 2.1.

$$R \leftarrow \alpha R + (1 - \alpha)M \tag{2.1}$$

R predstavlja povprečen čas RTT, M je RTT čas izmerjen za zadnji potrjeni paket, α je konstanta, ki pove koliko zadnja izmerjena vrednost vpliva na povprečno vrednost RTT. Priporočena vrednost za α je 0.9 (na ta način se v povprečni vrednosti R uporabi le 10% novo izmerjene vrednosti M). Za izračun časa za ponovno pošiljanje naslednjega paketa je še potrebno to vrednost pomnožiti z β, priporočena vrednost katere je 2 [1].

$$RTO = \beta R \tag{2.2}$$

Problem pri takem pristopu je konstantna vrednost odstopanja β, trdi Jacobson [1]. Zmnožek v 2.2 je potreben za zmanjšanje možnosti, da bo paket potreboval več časa, kot je izračunani povprečni RTT R v 2.1. V primeru velike obremenjenosti je možno, da je potreben čas še daljši od predvidenega. Po Jacobsonovih trditvah se ta vrednost prilagaja le omrežjem obremenjenim do 30%, torej, na obremenjenih omrežjih bo TCP ponovno pošiljal segmente, ki niso izgubljeni, ampak samo zamujajo. Na ta način se bo promet še povečal.

Jacobsonova rešitev vključuje v zgoraj navedene enačbe še deviacijo. Na začetku rešitve se enačba 2.1 zapiše v spremenjeni obliki:

$$R \leftarrow R + \alpha(M - R) \tag{2.3}$$

V tej enačbi M predstavlja izmerjen RTT čas, R je povprečen RTT čas, α je zaglajevalna konstanta, ponavadi okoli 0,1. Na levi strani R predstavlja oceno naslednjega merjenja, $M - R$ je napaka v tej oceni. Iz teh vrednosti se da izračunati povprečni čas RTT in njegovo deviacijo. Ker je za računanje standardne deviacije potrebno izračunati koren in kvadrat, sta se avtorja [1] odločila, da bo absolutna povprečna deviacija (angl. *absolute deviation* ali *mean deviation*) hitrejši, ampak dovolj dober približek. Naslednje funkcije prikazujejo izračun časa za ponovno pošiljanje z uporabo kvadratne deviacije.

$$Err \equiv M - R \tag{2.4}$$

$$R \leftarrow R + \alpha Err \tag{2.5}$$

$$V \leftarrow V + \beta(|Err| - V) \tag{2.6}$$

Enačba 2.6 računa povprečno deviacijo za naslednji približek. β je zaglajevalna konstanta, ki je ponavadi enaka 0.25. Na koncu se še izračuna vrednost za ponovno pošiljanje.

$$RTO = R + 4V \tag{2.7}$$

Takšen izračun RTO zmanjšuje verjetnost nepotrebnega ponovnega pošiljanja na prometnih omrežjih zaradi hitrejših in natančnejših prilagoditev.

2.2 Zamašitve v omrežju

Posamezni deli Internet omrežja so narejeni z uporabo različnih tehnologij. Komunikacija lahko poteka po delih omrežja z različno hitrostjo in pasovno širino. Po istih delih omrežja lahko komunicira več različnih komunikacijskih tokov istočasno. Lahko se zgodi tudi, da se zaradi napake prekine glavni komunikacijski kanal in se promet v celoti preusmeri po počasnejšem pomožnem kanalu. Zaradi te dinamičnosti dogodkov v omrežju je TCP narejen tako, da se dinamično prilagaja spremembam. Veliki problem predstavljajo točke kjer se več vhodnih kanalov povezuje v eden izhodni kanal na način, da je seštevek vhodnih pasovnih širin večji kot je izhodna pasovna širina. Taka točka se imenuje "ozko grlo" (angl. *choke point*) ker paketi lahko prihajajo hitrejše, kot jih je možno pošiljati naprej. V tem primeru čakalne vrste rastejo. Kadar se čakalne vrste do konca izpolnijo, prihajajoči paketi se zavržejo. Takšno stanje se imenuje zamašitev komunikacijskega kanala [7].

V primeru zamašitve se RTT čas poveča, veliko paketov pa sploh ne pride na cilj. Zaradi tega se lahko zgodi veliko število ponovnih pošiljanj, kar povzroča

še večjo zamašenost. Takšen sled dogodkov vodi do situacije znane pod imenom *zamašitveni kolaps* (angl. *congestion collapse*). Pri zamašitvenem kolapsu uporabna komunikacija skoraj da ne obstaja.

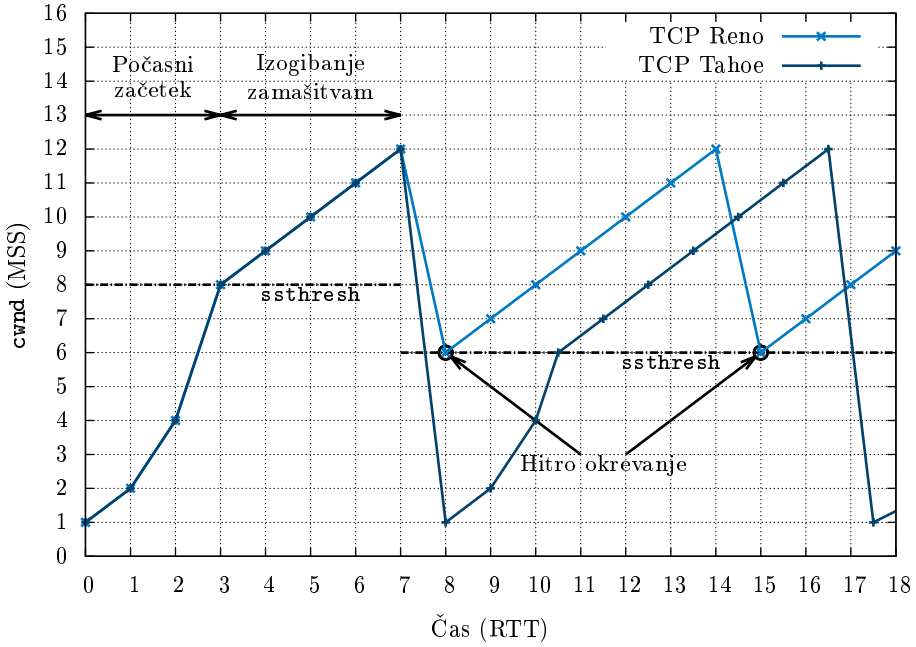
Jacobson je prvi raziskoval te tedaj še nepričakovane dogodke. Kot osnovo za rešitev tega problema je opisal princip "ohranjanja paketov" (angl. *"conservation of packets"*) [1]. Ta pojem je izposojen iz fizike in pomeni, da se število paketov v omrežju, ki je v ravnovesju, ohranja. Novi paket se ne vstavi v omrežje preden stari paket izide. Omrežje je v ravnovesju takrat, ko pošilja maksimalno število paketov stabilno in brez izgub. Omrežje s takšnimi lastnosti bi bilo robustno v primeru zamašitve. Obstajajo trije načini, da ta princip ne uspe:

- Povezava ne pride v ravnovesje. To pomeni, da pošiljatelj pošilja pakete prehitro ali prepočasno. Za časovno sinhronizacijo pošiljanja se lahko uporabijo potrditve paketov. Ker prejemnik pošilja potrditve z hitrostjo s katero paketi prihajajo, frekvenca prihodov potrditev določa hitrost pošiljanja. Da se povezava tako časovno sinhronizira, mora pošiljanje na začetku biti počasneje. Sčasoma ga je treba pospeševati do optimalne hitrosti. Ta faza se imenuje *počasni začetek* (angl. *slow start*).

- Pošiljatelj pošlje novi paket preden je stari paket sprejet. Ta problem se dogaja zaradi napačno definiranih časovnih merilcev za ponovno pošiljanje in je že opisan v odstavku 2.1.5.

- Ravnovesje se ne doseže zaradi omejenih sredstev za pošiljanje po poti. Pošiljatelj mora sprejemati signale o spremembah prometa po komunikacijskem kanalu, tako da lahko prilagodi hitrost pošiljanja. Ker pošiljatelj nikoli ne ve točno koliko pasovne širine je prosto za pošiljanje, mora počasi pospeševati hitrost pošiljanja, in na ta način preizkušati kanal za prosto pasovno širino. Ta faza se imenuje *izogibanje zamašitvam* (angl. *congestion avoidance*). Če se to ne bi izvajalo, lahko en del pasovne širine, ki ga je prej uporabljala nekatera druga povezava, ostane neuporabljen. Po drugi strani, pošiljatelj mora vedeti kdaj se bliža točka zamašitve. Ker je velika verjetnost da se je nekateri paket izgubil zaradi zamašitve, prej kot bi bil uničen pri prenosu po kanalu, je potekel čas za njegovo ponovno pošiljanje dobra indikacija zamašitve. Če se to zgodi mora pošiljatelj upočasniti pošiljanje, da se zmanjša gneča v čakalnih vrstah (spet se uporabi počasni začetek).

V nadaljevanju so zgoraj navedene rešitve opisane na primeru.



Slika 2.5: Prikaz spremembe `cwnd` okna za algoritme TCP Tahoe in TCP Reno. `ssthresh` je na začetku nastavljen na 8. Iz grafa je razvidno da TCP Reno pošilja pakete hitrejše kot TCP Tahoe zaradi preskočene faze počasnega začetka, in na ta način povečuje uporabljeni pretok.

2.2.1 Implementacija osnovnih algoritmov za preprečevanje zamašitev

TCP Tahoe in *TCP Reno* sta prvi različici TCP protokola z vgrajenimi algoritmi za preprečevanje zamašitev. Po ugotovljenih izgubah paketov TCP Reno poskuša uporabiti pakete v omrežju, ki še niso potrjeni, kot sistem za časovno sinhronizacijo, namesto da bi uporabil počasni začetek [7]. Na ta način se poveča pretok čez omrežje. Deluje na isti način kot TCP Tahoe z dodatkom algoritma *hitro okrevanje* (angl. *fast recovery*). V nadaljevanju sledi opis navedenih algoritmov [8], ki so grafično prikazani na sliki 2.5.

- Počasni začetek je narejen zaradi uspešne časovne sinhronizacije TCP protokola pri začetku komunikacije ali po ponovnem začetku, ki sledi prihajanju do točke zamašitve. Ta postopek dodaja novo spremenljivko TCP protokolu, poimenovano `cwnd` (angl. *congestion window*). Na ta način dva okna kontrolirata hitrost pošiljanja TCP protokola: okno,

objavljeno od strani prejemnika (opisano v odstavku 2.1.1) in okno, določeno pri pošiljatelju. Manjše od teh dveh oken določa velikost podatkov, ki jih TCP lahko v danem trenutku pošilja po omrežju. TCP ne sme poslati več kot toliko podatkov preden mora čakati na potrditev, da so podatki zunaj omrežja.

Na začetku povezave je `cwnd` nastavljen na velikost enega MSS, z vsakim prispelim ACK paketom se poveča za še en MSS. Na ta način se dejansko za vsako potrjeno pošiljanje `cwnd` okna njegova velikost podvoji (slika 2.3). Ta postopek traja dokler `cwnd` ne pride do vrednosti poimenovane *prag počasnega začetka* `ssthresh` (angl. *slow start threshold*) ali pa do izpada paketov. Začetna vrednost `ssthresh` je lahko poljubna, ampak je pogosto nastavljena na maksimalno vrednost 65535.

- Izogibanje zamašitvam sledi počasnem začetku, če še ni prišlo do izpadov paketov. V tej fazi se `cwnd` linearno povečuje za eden MSS pri potrditvi vseh podatkov iz trenutnega okna. Torej, če je `cwnd` enak pet, po uspešni potrditvi petih paketov se bo `cwnd` povečal na šest. Ta postopek traja do izpada paketov.

- Ker je načeloma možnost, da je paket uničen po poti zelo majhna, TCP pošiljatelj predpostavi, da se je paket izgubil zaradi zamašitve. V tem primeru je nujno upočasniti pošiljanje. Zaradi tega se po izgubi paketa `cwnd` zmanjša nazaj na velikost enega MSS, velikost `ssthresh` se zmanjša na polovico velikosti trenutnega `cwnd` okna. Potem se spet začne proces počasnega začetka.

- Preden se ponovno začne proces počasnega začetka, izpadli paket se mora ponovno poslati. Če kateri paket manjka, TCP pošilja ponovljene potrditve in na ta način pove nasprotni strani kateri paket še ni prišel na cilj. Namesto da bi se čakalo na potek časovnega merilca za ponovno pošiljanje se paket pošlje takoj po sprejetju treh ponovljenih potrditvah. Ta postopek se imenuje *hitro ponovno pošiljanje* (angl. *fast retransmit*). Ena ponovna potrditev lahko pomeni da paket zamuja, ampak po tri ponovne potrditve TCP lahko sklene da je paket izgubljen.

Hitro okrevanje (angl. *fast recovery*) je dodatni algoritem v TCP Reno protokolu, ki poskuša zmanjšati potrebo po ponovnem počasnem začetku po vsakem izpadu paketa. V primeru treh ponovnih potrditvah se `ssthresh` zmanjša na polovico poslanih in še nepotrjenih segmentov (torej segmentov, ki se še nahajajo v omrežju), `cwnd` se potem nastavi na vrednost `ssthresh` in zviša za

tri. Razlog, da se zviša za tri, so tri ponovljene potrditve pred hitrim ponovnim pošiljanjem. Vsaka potrditev pomeni, da je eden paket zapustil omrežje. Zaradi tega se lahko pošljejo tri dodatna nova paketa.

Vsaka dodatna ponovna potrditev zviša `cwnd` za ena, ker je še eden paket zapustil omrežje. TCP pošlje dodatni paket , če `cwnd` to dovoli.

Ko pride potrditev sprejetja novih podatkov se `cwnd` zmanjša na velikost `ssthresh` in se začne faza izogibanja zamašitvam. Ta potrditev potrjuje paket poslan s pomočjo hitrega ponovnega pošiljanja in vse ostale pakete med izgubljenim paketom in prihodom tretje ponovljene potrditve. V primeru, da takšna potrditev ne prispe preden poteče časovni merilec za ponovno pošiljanje, TCP začne z fazo počasnega začetka.

V primeru da je izguba paketa v TCP Reno protokolu namesto s tremi ponovljenimi potrditvami ugotovljena s pomočjo poteklega merilca za ponovno pošiljanje, namesto hitrega okrevanja se uporabi počasni začetek, torej v tem primeru deluje isto kot TCP Tahoe.

Ker se predpostavlja da se paketi izgubijo zaradi zamašitve, je iz navedenih opisov razvidno, da ta protokola nista primerna za povezave z izgubami. V takem primeru se izgubljeni paketi lahko napačno obravnavajo kot paketi izgubljeni zaradi zamašitve in ne zaradi uničenosti podatkov. To lahko drastično vpliva na pretočnost podatkov zaradi nepotrebnih zmanjšanj hitrosti pošiljanja.

Poglavje 3

Izkoriščanje pasovne širine gigabitnih omrežij

3.1 Pohitritev omrežij

Začetne tehnologije so uporabljale električne impulze za prenos podatkov po omrežjih. Razvoj gigabitnih omrežij je ozko povezan z razvojem optičnih vlaken [9]. V obeh primerih (razširjanje elektronov po prevodniku in razširjanje fotonov po optičnem vlaknu) je hitrost širjenja signalov približno enaka. Razlika je v tem, da so v optičnem vlaknu biti manjši. To pomeni, da se v eni časovni enoti pošlje več bitov v optičnem vlaknu kot v bakreni žici. Vrednost, ki določa število bitov, ki se lahko pošljejo v eni časovni enoti, se imenuje *pasovna širina* (angl. *bandwidth*). Ponavadi se označuje z enoto `bps` ali `b/s` (število bitov v sekundi ali angl. *bits per second*).

Zakasnitev ali *latenca* (angl. *delay* ali *latency*) je čas, potreben za prenos paketa. Zakasnitev se ponavadi meri v obeh smereh (angl. *round-trip time latency*) s pomočjo programa *ping*. Tako izmerjena zakasnitev je čas od pošiljanja paketa na enem računalniku do sprejema odgovora na ta paket na istem računalniku. Zakasnitev v eni smeri je čas, potreben za prehod paketa od pošiljatelja do prejemnika po omrežju (angl. *end-to-end latency*).

Na zakasnitev vpliva več različnih dejavnikov. Dva najpomembnejša dejavnika sta čas, potreben za širjenje signala (angl. *propagation delay*) in čas za pošiljanje (angl. *transmission delay*) [2]. Čas za pošiljanje je čas, ki ga naprava za komunikacijo rabi za prenos vseh bitov paketa na prenosni medij in je odvisen od pasovne širine medija. Čas za širjenje signala je odvisen od hitrosti signala in dolžine medija, oziroma razdalje med pošiljateljem in prejemnikom v primeru brezžičnih povezav.

$$BDP = Bandwidth * Delay \tag{3.1}$$

BDP zmnožek (angl. *bandwidth delay product*) je zmnožek latence, merjene v eni smeri, in pasovne širine. Določa največje število bitov, ki se trenutno nahajajo v omrežju (ki so poslani in še nepotrjeni). Iz enačbe 3.1 je razvidno, da povečevanje latence ali pasovne širine povzroča povečevanje vrednosti BDP. Omrežja z visokim produktom BDP imajo angleško ime *long fat networks* ali skrajšano LFN.

Leta 2007 je *Institute of Electrical and Electronics Engineers* (skrajšano IEEE) predlagal izdelavo hitrih 40 Gb/s in 100 Gb/s omrežjih [10]. Sestavljena je delovna skupina specializirana za raziskovanje in izdelavo standardov za takšna omrežja. Projekt se je končal leta 2010. V letu 2011 je izdelano nekaj uporabnih različic omrežja, narejenih po teh standardih. Osnovna zahteva novih omrežjih je delovanje v skladu z 802.3 Ethernet standardi, določene so tudi razdalje, na katerih povezava mora delovati pri različnih prenosnih medijih. Obstoječe povezave z hitrostjo 10 Gb/s ali 25 Gb/s se kombinirajo v eno povezavo z več pasov ali valovnih dolžin, tako se doseže želena hitrost.

3.2 Problemi pri LFN

Z uvedbo optičnih vlaken je hitrost prenosa podatkov začela hitro naraščati. Začetne različice TCP protokola ne zmorejo več izkoristiti vso pasovno širino. Težave z uporabo TCP protokola pri takih omrežjih z velikim BDP produktom se lahko razvrstijo v dve kategoriji: zmogljivost in zanesljivost [11].

Na zmogljivost omrežja ne vpliva hitrost prenosa sama, ampak produkt hitrosti prenosa in RTT, torej BDP produkt. Kot je že povedano, BDP produkt določa število bitov v omrežju na način, da je omrežje popolnoma uporabljeno. Tri problema nastajajo pri uporabi začetnih različic TCP protokola na LFN omrežjih.

Prvi problem je šestnajst-bitna velikost sprejemnega okna, definiranega v poglavju 2.1.1. To pomeni da v danem trenutku ena TCP povezava lahko pošilja maksimalno 2^{16} oziroma 65,535 bajtov. Zaradi tega v primerih ko je BDP produkt večji od 65535 bajtov, pasovna širina omrežja ni popolnoma uporabljena. Za rešitev tega problema se uporablja dodatna TCP možnost poimenovana merilo okna (angl. *window scale*) [11]. Končna velikost okna se izračuna tako, da se velikost okna zamakne v levo stran za število, definirano v možnosti merila okna. Ta operacija je identična množenju z potenco števila 2. V primeru da okno v TCP zaglavju iznaša 10000 bajtov, TCP možnost merila

okna iznaša 3, končno okno dobi velikost $10000 * 2^3 B$. Maksimalni zamik v levo stran iznaša 14 mest.

S takšnim povečevanjem okna se povečuje tudi verjetnost izgub paketov v omrežju. Začetne različice TCP protokola so izpraznile pretok podatkov pri vsaki izgubi. Po drugi strani, linearno povečevanje okna (angl. *additive increase / multiplicative decrease* ali skrajšano AIMD) potrebuje veliko časa, da se pasovna širina ponovno popolnoma izkoristi. Selektivno potrjevanje (angl. *selective acknowledgement* ali skrajšano SACK) je algoritem, ki izboljšuje algoritme za hitro ponovno pošiljanje in hitro okrevanje pri LFN omrežjih. V primerjavi z navadnimi potrditvami, SACK prikazuje popolno stanje potrjenih paketov in paketov, ki še niso prišli do prejemnika.

Večje izgube pomenijo slabše rezultate pri RTT meritvah in ocenjevanju naslednjega časa RTO [12]. Potrditev ponovno poslanega paketa se ne sme uporabiti pri ocenjevanju RTO, ker se ne ve če ta potrditev potrjuje originalni paket ali njegovo ponovno poslano kopijo. Večina TCP implementacij meri čas za ocenjevanje RTO za samo eden paket v trenutnem oknu. Pri uporabi velikih oken napake pri prenosu pomenijo redkejše popravke RTO. Zaradi tega se v novejših različicah TCP uporabljajo časovne oznake (angl. *timestamps*). TCP protokol pošiljatelja doda možnost v TCP zaglavje, ki vsebuje čas pri pošiljanju paketa. Prejemnik zatem pri sprejemu prepiše to časovno oznako v paket, ki potrjuje sprejem tega paketa. RTT čas se izračuna z enostavnim odštevanjem trenutnega časa pri sprejemu potrditve in časovne oznake, zapisane v TCP zaglavju potrditve. Potem se lahko z uporabo RTTM algoritma oceni novi čas RTO.

Problemi z zanesljivostjo TCP protokola pri LFN omrežjih se večinoma nanašajo na nenamerno ponovno uporabo števila sekvence. V tem primeru se lahko napačni segment sprejeme kot veljavni segment v povezavi. Ta problem se lahko zgodi v dveh primerih.

- V trenutni povezavi se zaradi hitrosti uporabijo vsa možna števila sekvence (kot je že povedano, števila sekvence se spreminjajo po modulu 2^{32}). Prejemnik potem lahko sprejeme dva različna paketa z istim številom sekvence, če je eden izmed teh dveh paketov zamujal.

- Če se povezava zaključi in zatem odpre nova z istim parom vtičnic, se lahko zgodi, da paket iz prejšnje zaključene povezave zaradi zamud pride kot veljaven paket v novo povezavo.

PAWS algoritem (angl. *Protect Against Wrapped Sequence numbers*) uporablja iste časovne oznake kot izboljšani RTTM algoritem. S pomočjo pri-

merjanja časovnih oznak paketov se lahko ločita točen paket in njegov starejši dvojnik.

Dodaten problem, ki nastaja pri vseh različicah TCP protkolov, je poštenost (angl. *fairness*) oziroma prijaznost (angl. *friendliness*). To se nanaša na situacije, v katerima usmerjevalnik usmerja več različnih podatkovnih tokov. Vprašanje je, če določeni TCP protokol omogoča pravično porazdelitev dostopne pasovne širine ali pa za svojo povezavo požrešno uporablja čim več, ne da bi dovolil uporabo ostalima povezavama. Vsak algoritem poskuša rešiti ta problem na drug način.

Poglavje 4

Simulatorji in simulacije

4.1 Splošno o modelih in simulacijah

Modeliranje je pomembno pri načrtovanju in razvoju sistemov. Uporablja se za poenostavljeni prikaz dejanskega sistema. Sistem se lahko prikaže na dva načina: s pomočjo analitičnega pristopa in s pomočjo simulacij. Analitični pristop vključuje matematična orodja in pripomočke. Simulacijski pristop se uporablja pri večjih sistemih, kjer matematični pristop ni mogoč [18].

Pri simulaciji se razvija model dejanskega sistema z namenom izvajanja poskusov v namen razumevanja obnašanja in delovanja tega sistema. Priporočeno je da se za definiranje problema in načrtovanje modela in poskusov uporabi 40% časa, 20% časa se uporabi za programiranje tako določenega modela. Na koncu se 40% časa uporabi za preverjanje modela, dodatne poskuse in analizo rezultatov [19].

V nadaljevanju so opisani osnovni deli vsake simulacije [18].

- *Entitete* so objekti, ki vplivajo ena na drugo in spreminjajo stanja med seboj. Na ta način prinašajo spremembe stanja sistema. Različne vrste entitet vsebujejo različne atribute. Primeri entitet so računalniki, usmerjevalniki in paketi.

- *Sredstva* se uporabljajo za izvajanje simulacije in so ponavadi dostopna v omejenih količinah. Zaradi tega jih entitete morajo deliti med seboj. Primera sredstev sta pasovna širina in čas za oddajanje signalov.

- *Dogodki* se ustvarijo kadar se entiteta vključi v nekatero *aktivnost*. Takšno delovanje v aktivnostih povzroča spremembe stanja sistema. Primeri ak-

tivnosti so čakanje na dostopna sredstva (v primeru, da so vsa sredstva zasedena) in čakanje v vrstah.

- *Razporejevalnik* hrani seznam dogodkov in časov za izvajanje. Med izvajanjem simulacije ustvarja dogodke in jih izvaja.
- *Globalne spremenljivke* hranijo podatke, dostopne celotni simulaciji.
- *Tvorec naključnih številk* uvaja naključnost v simulacije. Ponavadi se številke izračunajo s pomočjo različnih algoritmov za naključna števila, ampak se zdi, da so naključno izbrane (takšen tvorec naključnih števil se po angleško imenuje *pseudo-random nuber generator*). Ker je pogosto naslednja naključna številka odvisna od prej izračunate naključne številke, se s spremembo začetne številke (angl. *seed*) spremeni celotno zaporedje naključno izbranih številk. Na ta način se lahko ista simulacija požene večkrat z različnimi začetnimi številkami in se dobijo različni statistični podatki.
- *Zbiralec statističnih podatkov* zbira podatke med izvajanjem simulacije. Po končani simulaciji se ti podatki lahko uporabijo za raziskovanje in načrtovanje.

Simulacije so pogosto odvisne od časa. Takšne simulacije hranijo časovni merilec, ki ustavi izvajanje simulacije po določenem poteklem času. Obstajata dve vrsti simulacij odvisnih od časa: simulacije, ki jih poganjajo časovni intervali in simulacije, ki jih poganjajo dogodki. Prva vrsta simulacij izvaja dogodke vsak določeni časovni interval, medtem ko druga vrsta simulacij lahko pripravi in izvede dogodke kadarkoli.

4.2 Simulacijski programi

Danes obstaja veliko dostopnih plačljivih in brezplačnih programov za izdelavo simulacij omrežja. V nadaljevanju je našteto in na kratko opisano nekaj različnih programov za omrežne simulacije.

4.2.1 NS3

NS3 [13] je naslednik starejšega simulatorja NS2. Narejen je kot samostojni projekt, ne kot nadgradnja NS2. Zaradi tega ne podpira simulacij, narejenih za NS2.

Simulacija se lahko razvije popolnoma z uporabo programskega jezika C++. Modularnost je narejena podobno kot pri NS2. Moduli so narejeni v programskem jeziku C++. Uporabnik po želji lahko simulacijo definira v jeziku Python ali v jeziku C++. NS3 se še zmeraj aktivno razvija, zaradi tega nekatere komponente še niso implementirane. Po drugi strani omogoča dodatne sodobne tehnologije, kot je prikaz simulacije v "pcap" obliki. Dodane so in spremenjene nekatere možnosti iz NS2, kot so IP naslavljanje in izboljšana uporaba več različnih vmesnikov na enem vozlišču.

4.2.2 OMNeT++

OMNeT++ [14] je modularno razvojno okolje za izdelavo omrežnih simulacij na bazi interaktivnega razvojnega okolja (angl. *interactive development environment* ali skrajšano IDE) Eclipse.

Moduli so programirani v programskem jeziku C++. Pri izdelavi simulacije se moduli združijo v omrežno arhitekturo s pomočjo visoko-nivojskega programskega jezika, poimenovanega Ned. Potem ko je struktura simuliranega omrežja definirana, določijo se parametri simulacije znotraj konfiguracijske datoteke. Nazadnje se simulacija izgradi in požene. Rezultati so shranjeni v dve obliki: v eni datoteki so vektorski zapisi, v drugi so skalarni zapisi. Obe datoteki se lahko uporabita v drugih programih za predstavitev rezultatov, kot sta R in Matlab.

OMNeT++ se lahko poganja na sistemih Linux, Unix, Windows in Mac OS X. OMNeT++ ima tudi trgovinsko različico, poimenovano OMNEST.

4.2.3 Opnet Modeler

Modeler [15] je komercialen simulator računalniških omrežjih, narejen za testiranje in prikaz novih tehnologij v realističnih scenarijih. Vsebuje urejevalnike za več različnih nivojev simulacije. Glavni urejevalnik se uporablja za postavitev vseh vozlišč simuliranega omrežja (viri, usmerjevalniki, antene in tako naprej), katerih delovanje se potem definira z uporabo urejevalnika vozlišč. V vozliščih se dogajajo različni procesi. Na najnižji ravni je urejevalnik procesov, ki definira avtomate prehajanja stanj znotraj procesa.

Vsebuje veliko grafičnih prikazov za lažjo uporabo. Lahko se poveže na zunanje objekte, datoteke in druge simulacije. Deluje na sistemih Windows in Linux.

4.2.4 Tetcos NetSim

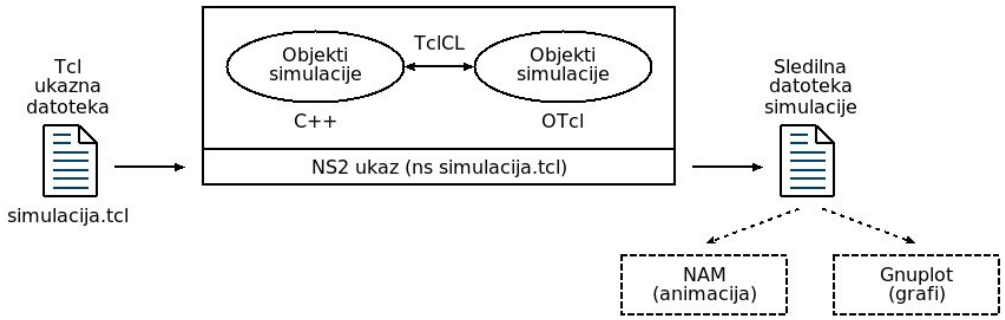
NetSim [16] je orodje za raziskavo in simuliranje omrežjih. Narejen je kot pripomoček pri učenju študentov o računalniških omrežjih. Razen možnosti simuliranja, NetSim ponuja vaje iz programiranja v jeziku C/C++, animacije, ki prikazujejo osnove delovanja računalniških omrežjih, in zbiranje in prikaz podatkov (paketov) iz obstoječih realnih omrežjih. Uporabniki lahko spreminjajo obstoječe knjižnice v C urejevalniku in programirajo nove knjižnice. NetSim ponuja tudi možnost razhroščevanja.

Obstajata dve različici NetSim orodja: akademska in standardna. Netsim se lahko uporablja na sistemih Windows.

4.3 Network Simulator 2 (NS2)

NS2 [17] je odprtokodno orodje za simulacijo računalniških omrežij. Njegovo delovanje je pogojeno spremembi dogodkov znotraj simulatorja. Lahko simulira vse omrežne funkcije in protokole s pomočjo dodatnih in po meri narejenih modulov. Veliko ljudi ga uporablja in nadgrajuje zaradi njegove fleksibilnosti in uporabnosti [18].

Slika 4.1 predstavlja osnovno arhitekturo simulatorja NS2. Sestavljata ga dva ključna dela. Interni mehanizem definira delovanje objektov simulatorja, narejen je v programskem jeziku C++. Simulacija je sestavljena v obliki ukaznih datotek v jeziku *Object-oriented Tool Command Language* (OTcl). Na ta način se simulacije pri manjših spremembah hitrejše izvajajo ker ni potrebno simulacijo prevajati (OTcl ukazne datoteke se ne prevajajo v strojno kodo, temveč se direktno izvajajo).



Slika 4.1: Osnovna arhitektura simulatorja NS2 [18]

Po uspešno končani simulaciji, NS2 shrani vrednosti v datoteko z zapisi. Iz te datoteke se potem lahko naredijo grafi simulacije ali grafični prikaz prehodov paketov. Orodje za prikaz animacije se imenuje Network Animator (NAM), za grafe se lahko uporabi GNUPlot.

Simulacije v NS2 se dogajajo v treh korakih. Prvi korak je zasnova simulacije. V tej fazi se določa smisel simulacije, načrt računalniškega omrežja, ki se bo simuliralo, ter tip predvidenih rezultatov. Drugi korak je sestavljen od dveh delov. V prvem delu se določajo vrednosti različnih komponent ter dogodki, ki se bodo izvedli v določenem času. V drugem delu tega koraka se tako postavljena simulacija zažene. Ko se simulacija konča, nastopa tretji korak, v kateremu se pregledujejo rezultati simulacije v namenu določanja napak simuliranega elementa in določa zmogljivost omrežja.

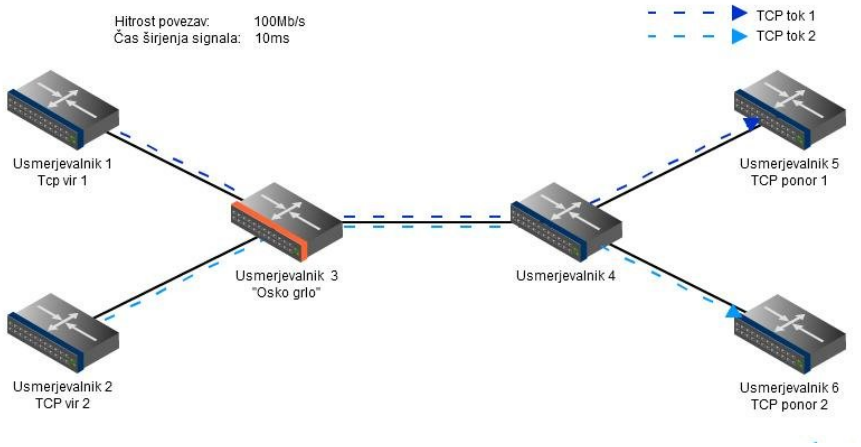
Vse simulacije v diplomski nalogi so narejene z uporabo simulatorja NS2. Čeprav je starejši simulator, NS2 ima veliko podporo skupnosti. Veliko različnih modulov in informacij je prosto dostopno na Internetu.

Poglavje 5

Simulacije TCP protokolov

5.1 Simulacijski model

Simulacije so narejene na enostavnem modelu, prikazanem na sliki 5.1. Model predstavlja "ozko grlo" v omrežnem prometu. TCP podatki prihajajo iz smeri usmerjevalnikov 1 in 2 ter čez usmerjevalnik 3 odhajajo v smeri usmerjevalnika 4. Ker usmerjevalnik lahko sprejeme več podatkov kot jih odda, TCP pošiljatelja morata kontrolirati hitrost pošiljanja.



Slika 5.1: Simulacijski model, na katerem so izpeljane vse simulacije.

Vse povezave imajo isto pasovno širino in čas širjenja signala. Pasovna širina je nastavljena na 100 Mb/s, čas, potreben za širjenje signala je 10 ms. Vrsta na "ozkem grlu" lahko primi 800 paketov, deluje kot *DropTail* vrsta

(to pomeni da se v primeru polne vrste vsi prihajajoči paketi zavržejo). Okna ponorov so nastavljena na vrednosti dovolj velike da se uporabi celotna pasovna širina. Vse simulacije so narejene na intervalu 500 sekund.

Na začetku sta simulirana dva osnovna protokola, TCP Tahoe/Reno, kot predstavnik algoritmov, ki delujejo na bazi izgubah, in TCP Vegas kod predstavnik algoritmov, ki delujejo na osnovi zakasnitvah v vrstah. Potem še sledijo algoritmi, narejeni za LFN povezave: TCP BIC/CUBIC, TCP Hybla in Compound TCP.

Simulacije prikazujejo pomembne lastnosti TCP protokolov.

- Prva simulacija prikazuje spremembe zamašitvenega okna `cwnd`. To velikost hrani vsak TCP pošiljatelj. Na določenih časovnih intervalih se ta velikost shrani v zasebno datoteko. V tem primeru je najboljša vrednost za časovni interval čas RTT, ker se `cwnd` spreminja za vsak RTT.

- Druga simulacija prikazuje zasedenost čakalne vrste. Deluje podobno kod prva simulacija. Vklopljeno je opazovanje vhodne vrste. Ta objekt hrani število bajtov in paketov v čakalni vrsti čez celotno simulacijo. V določenih časovnih intervalih se število bajtov `size_` shrani v zasebno datoteko in prikaže na grafu.

- Tretja simulacija prikazuje izgubljene pakete pri zamašitvah. Narejena je na isti način kot druga simulacija, ampak uporabljena spremenljivka prikazuje število izgubljenih paketov v čakalni vrsti `p_drops_`.

- Četrta simulacija prikazuje zakasnitve v čakalni vrsti. Na začetku so se zakasnitve v vrsti računale s pomočjo ukazne datoteke, narejene v jeziku *Python* iz sledilne datoteke. Pokazalo se je da pri večjih simulacijskih modelih sledilne datoteke lahko dosežejo nekaj Gigabajtov prostora. Zaradi tega je spremenjen način računanja zakasnitev tako, da je spremenjena izvirna koda simulatorja, in sicer objekta `Queue/DropTail`. Za vsako vrsto se lahko vklopi nov način shranjevanja pomembnih podatkov, ki je zmanjšan samo na nujne podatke. Iz teh podatkov se lahko preberejo prihodi in odhodi paketov iz vrste in na ta način se dobi čas čakanja paketa v vrsti.

- Peta simulacija prikazuje RTT čase. Računanje RTT časov je še eden primer, ki je na začetku bil izdelan iz sledilne datoteke in potem spremenjen v originalni kodi simulatorja. Spremenjena sta objekta `Agent/TCP` in `Agent/TCPSink`. Za vsaki od teh objektov je narejeno shranjevanje

odhodov in prihodov paketov. Zatem se s pomočjo unikatnih oznak paketov izračuna RTT čas za vsak paket posebej.

- Šesta simulacija prikazuje uporabljeno pasovno širino. Na ponorih se za vsak določeni interval dobi število sprejetih bajtov od vsakega vira posebej. To število se podeli s časovnim intervalom in se na ta način dobi uporabljena pasovna širina za enega pošiljatelja. Število sprejetih bajtov se po vsakem intervalu nastavi na 0.

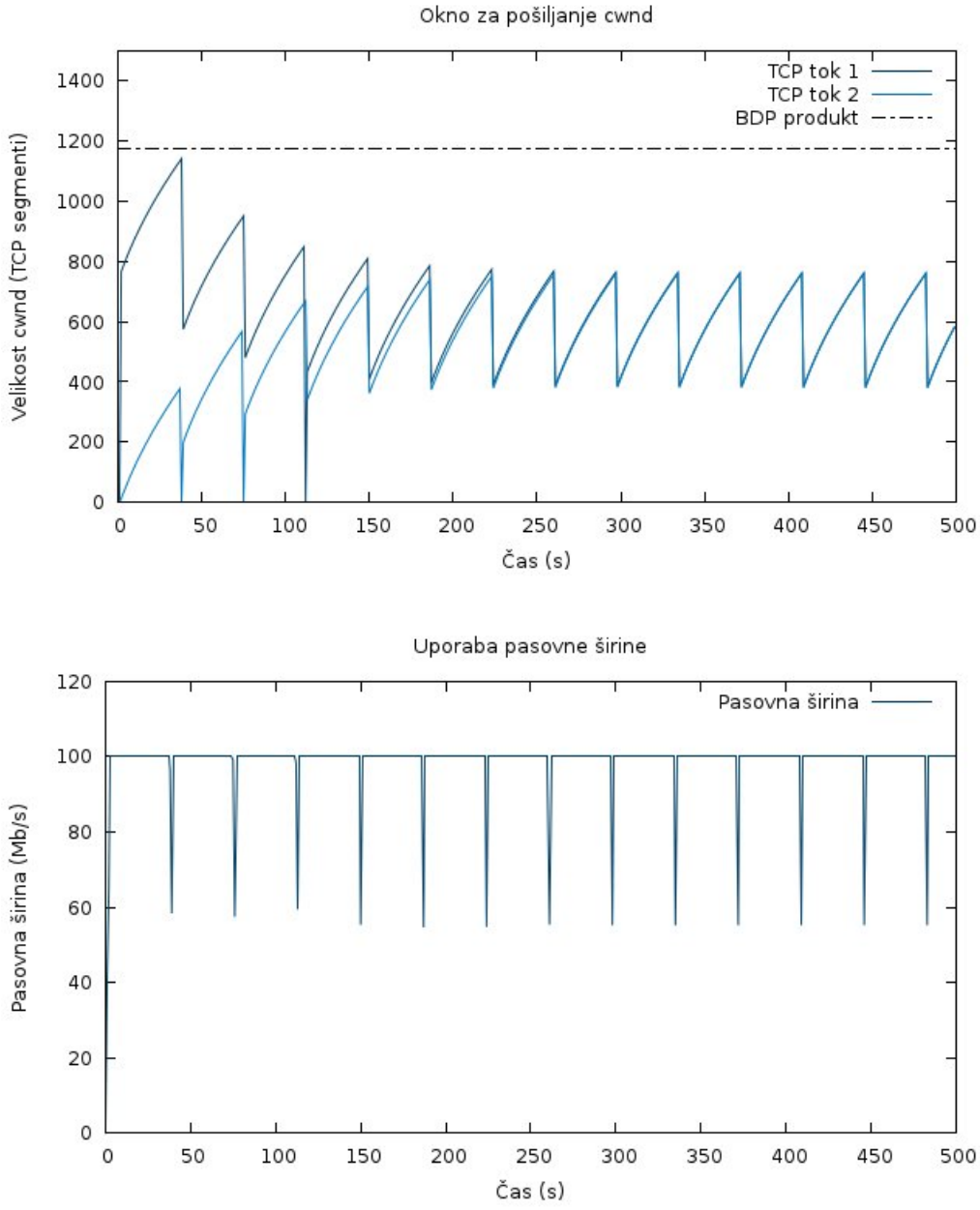
5.2 TCP Tahoe in TCP Reno

TCP Tahoe in TCP Reno sta prvi različici TCP protokola, ki poskušata preprečiti zamašitveni kolaps v omrežjih. Nastala sta na koncu osemdesetih. Zasnovana sta izključno na izgubah paketov v omrežju. Njuno delovanje je že opisano v poglavju 2.2.1. Grafi prikazujejo rezultate pridobite s simulacijo TCP Reno protokola, zaradi njegovega boljšega delovanja na LFN omrežjih.

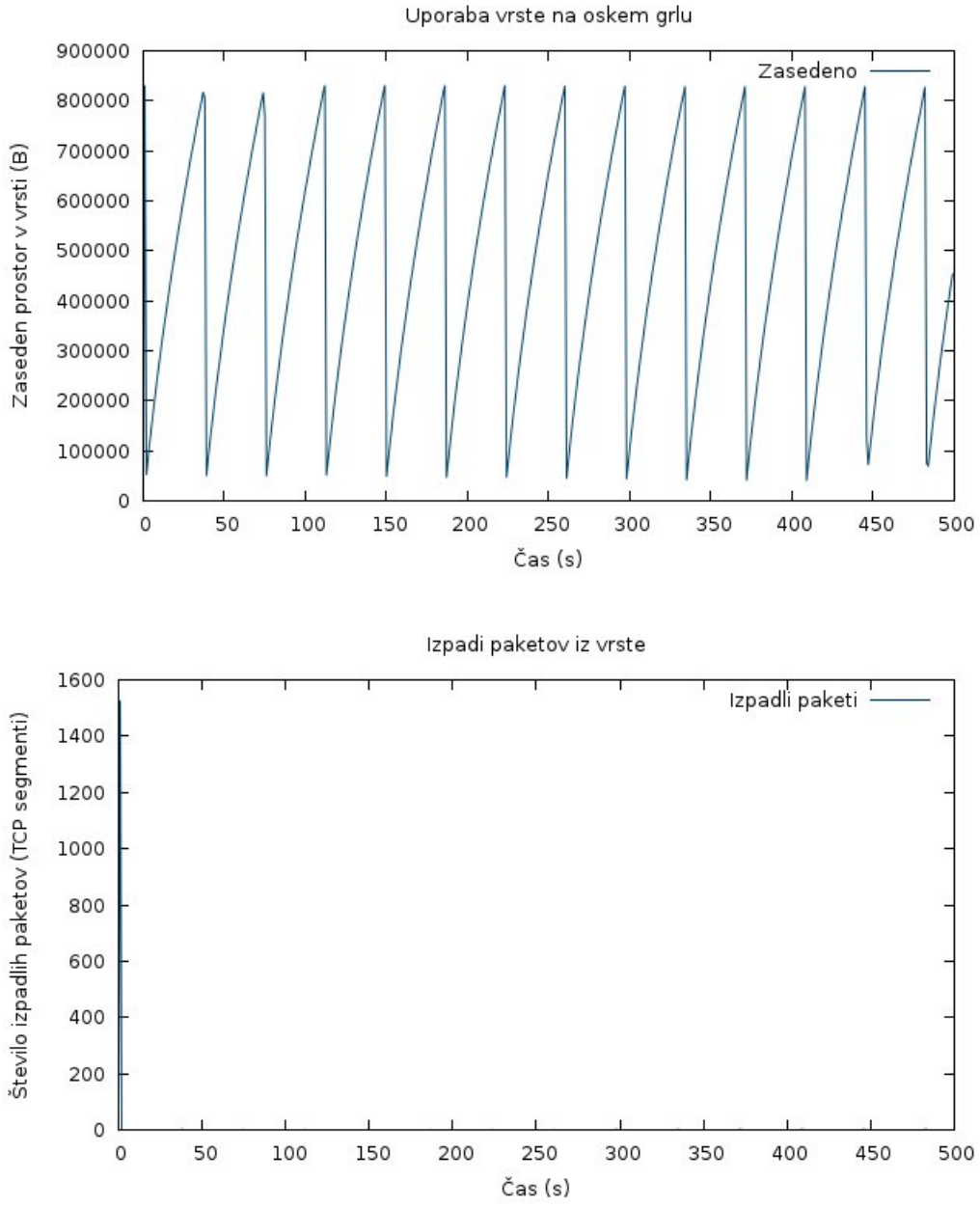
5.2.1 Rezultati simulacije TCP Reno protokola

Kot je že povedano, simulacija najstarejših TCP algoritmov za izogibanje zamašitvam je narejena kot referenca za naslednje simulacije. Največji problem je AIMD način delovanja algoritma. Ker se v fazi izogibanja zamašitvam okno za pošiljanje *cwnd* povečuje samo za en segment v enem RTT, pri hitrih LFN omrežjih je lahko potrebno veliko časa da se popolni celotno okno. Kadar pride do izgube, okno se multiplikativno zmanjša. Graf izpadlih paketov 5.2.1 prikazuje veliko število izgubljenih paketov na samem začetku povezave. Vsakič ko se zgodi tudi majhen izpad paketov, okno za pošiljanje se zmanjša, in s tem tudi uporaba pasovne širine, prikazana na grafu 5.2.1. Zasedenost vrste je kar velika ker se paketi nabirajo dokler se ne zazna izguba, potem se vrsta do konca izprazni. Algoritem potem rabi veliko časa da pretok ponovno naraste.

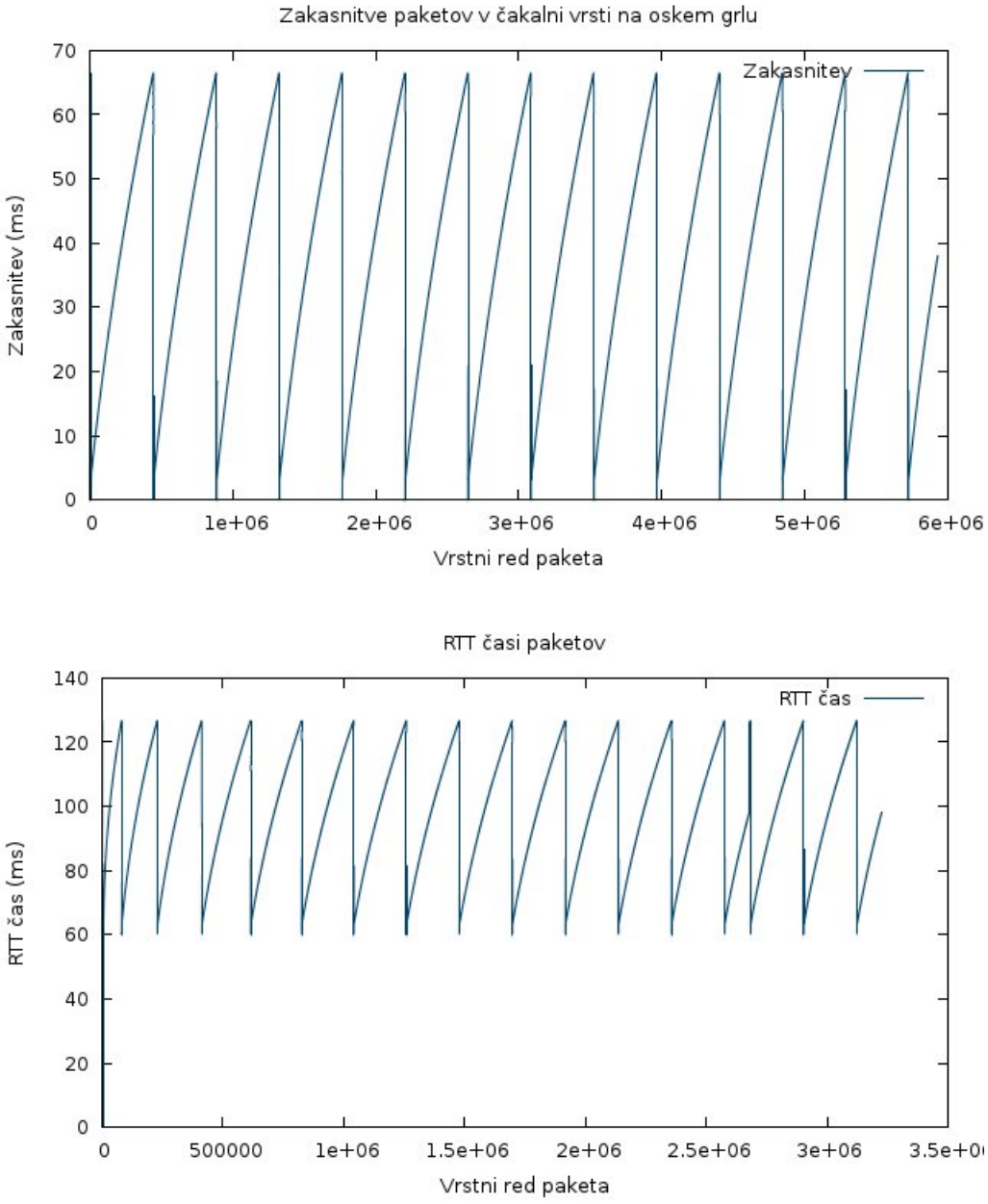
Pri algoritmih Reno oziroma Vegas je možna velika začetna vrednost spremenljivke `ssthresh`. Ker v fazi počasnega začetka algoritem hitro eksponentno narašča do praga `ssthresh`, lahko se zgodi da velikost poslanih podatkih gre čez mejo. Pri LFN omrežjih je lahko število izpadlih paketov kar veliko. Vsi izpadli paketi se morajo ponovno poslati. Ta korak je nujen za oceno prostega pretoka. V primerih ko je `ssthresh` majhen, faza izogibanja zamašitvam se začne prezgodaj. V nadaljevanju bo razvidno iz simulacij da tudi sodobnejši algoritmi uporabljajo to tehniko določanja uporabnega pretoka.



Slika 5.2: Okno za pošiljanje in pasovna širina TCP Reno protokola



Slika 5.3: Izkoriščenost in izpadi paketov iz vrste TCP Reno protokola



Slika 5.4: Zakasnitve v vrsti in RTT čas paketov TCP Reno protokola

5.3 TCP Vegas

TCP Vegas [20] je različica TCP protokola, narejena za prikaz novega pristopa do preprečevanja zamašitev. Algoritem uporablja tri tehnike za boljši pretok podatkov in manjše izgube.

5.3.1 Ponovno pošiljanje pri algoritmu TCP Vegas

Vegas uporablja in dograjuje osnovna algoritma, ki jih uporablja TCP Reno. Prvi algoritem je ponovno pošiljanje po poteklem časovnem števcu za ponovno pošiljanje, drugi je ponovno pošiljanje po treh ponovljenih potrditvah zadnjega uspešno sprejetega segmenta pred izgubo. Vegas shranjuje sistemski čas vsakič, ko je segment poslan. Pri prihodu potrditve se sistemski čas ponovno prebere. Zaradi pogostejših meritev je RTT približek pravilnejše določen. Vegas ponovno pošilja v dveh dodatnih primerih:

- Kadar pride dvojna potrditev, Vegas preveri, če je razlika med trenutnim časom in shranjeno časovno oznako večja kot dovoljena vrednost za čakanje pred ponovnim pošiljanjem. Če je, Vegas ponovno pošlje segment in ne čaka na dodatne ponovne potrditve.
- Če pride navadna potrditev in če je prva ali druga po ponovnem pošiljanju, Vegas naredi isto kot v prejšnji točki. To pohitri ponovno pošiljanje vseh ostalih segmentov, ki so morebitno izgubljeni pred ponovnim pošiljanjem, brez čakanja na ponovne potrditve.

Vegas preverja potek časa za ponovno pošiljanje samo občasno, tako da delovanje algoritma ni obremenjeno.

5.3.2 Preprečevanje zamašitev pri algoritmu TCP Vegas

Vegas preverja spremembe pri hitrosti pošiljanja podatkov. Primerja izmerjene velikosti z predvideno velikostjo pretoka. Hitrost pošiljanja ne sme biti večja kot razpoložljiv pretok, drugače se paketi hranijo v čakalnih vrstah na vmesnem usmerjevalniku z “ozkim grlom”. Cilj Vegasa je ohranjati želeno velikost takih dodatnih podatkov. Razen zavrženih paketov Vegas uporablja tudi spremembe velikosti odvečnih podatkov v omrežju. Algoritem preprečevanja zamašitev ne deluje v fazi počasnega začetka.

Algoritem deluje v nekaj korakih.

- V prvem koraku za trenutno povezavo algoritem določi vrednost `BaseRTT` kot najmanjšo izmerjeno vrednost RTT. Pogosto je to RTT vrednost prvega poslanega segmenta. Potem se izračuna *pričakovani pretok* z uporabo enačbe 5.1, kjer sta `Expected` pričakovana velikost pretoka in `WindowSize` velikost trenutnega okna za pošiljanje.

$$Expected = WindowSize/BaseRTT \tag{5.1}$$

- Drugi korak določi *dejansko hitrost pošiljanja*, označeno `Actual`. Ker za vsak poslani segment TCP Vegas hrani čas pošiljanja in čas prihoda potrditve, dejansko hitrost pošiljanja je možno izračunati tako, da se podeli število poslanih bajtov med časom pošiljanja segmenta in sprejemom njegove potrditve, in RTT čas segmenta. Ta izračun se naredi enkrat na RTT cikel.

- V tretjem koraku se primerjajo pričakovani pretok in dejanska hitrost pošiljanja. Enačba 5.2 določa diferencial, ki se uporablja pri računanju sprememb velikosti okna za pošiljanje. Diferencial je zmeraj enak ali večji kot nič. Enačba 5.3 določa zgornji in spodnji prag, ki določata preveč oziroma premalo dodatnih podatkov v omrežju.

$$Diff = Expected - Actual \tag{5.2}$$

$$\alpha < Diff < \beta \tag{5.3}$$

Kadar je $Diff < \alpha$, okno za pošiljanje se linearno povečuje, ker je velikost dodatnih podatkov v omrežju premajhna in zaradi tega razpoložljiva pasovna širina ni popolnoma uporabljena. Kadar je $Diff > \beta$, v omrežju je preveč dodatnih podatkov in nastopa zamašitev. Zaradi tega se okno za pošiljanje linearno zmanjšuje. V primeru, da je $\alpha < Diff < \beta$, okno ostane iste velikosti. V tem območju je pasovna širina najboljše uporabljena.

Ker števili α in β predstavljata meji pri uporabi pasovne širine, njihovi vrednosti sta prikazani v enoti kB/s. Izbrani sta odvisno od velikosti `BaseRTT`, zaradi tega je točnost celotnega algoritma odvisna od točnega izračuna te številke.

5.3.3 Počasni začetek pri algoritmu TCP Vegas

Obnašanje prvega počasnega začetka je različno od obnašanja vseh nadaljnjih počasnih začetkih. Pri izvajanju prvega počasnega začetka ne obstajajo nobeni

podatki, ki bi določili točno ali približno dostopno pasovno širino. Ko se počasni začetek zgodi med komunikacijo, algoritem ima dostop do velikosti okna za pošiljanje, pri katerem so se zgodile izgube paketov.

Če je začetna vrednost praga majhna, eksponentno povečevanje se bo ustavilo prezgodaj. Potem bo povezava rabila veliko časa da se popolnoma izpolni celotna razpoložljiva pasovna širina. Če je začetna velikost praga prevelika, algoritem bo poslal preveč podatkov prehitro. Zaradi tega bo prišlo do izgub. Te izgube so večje pri hitrejših omrežjih.

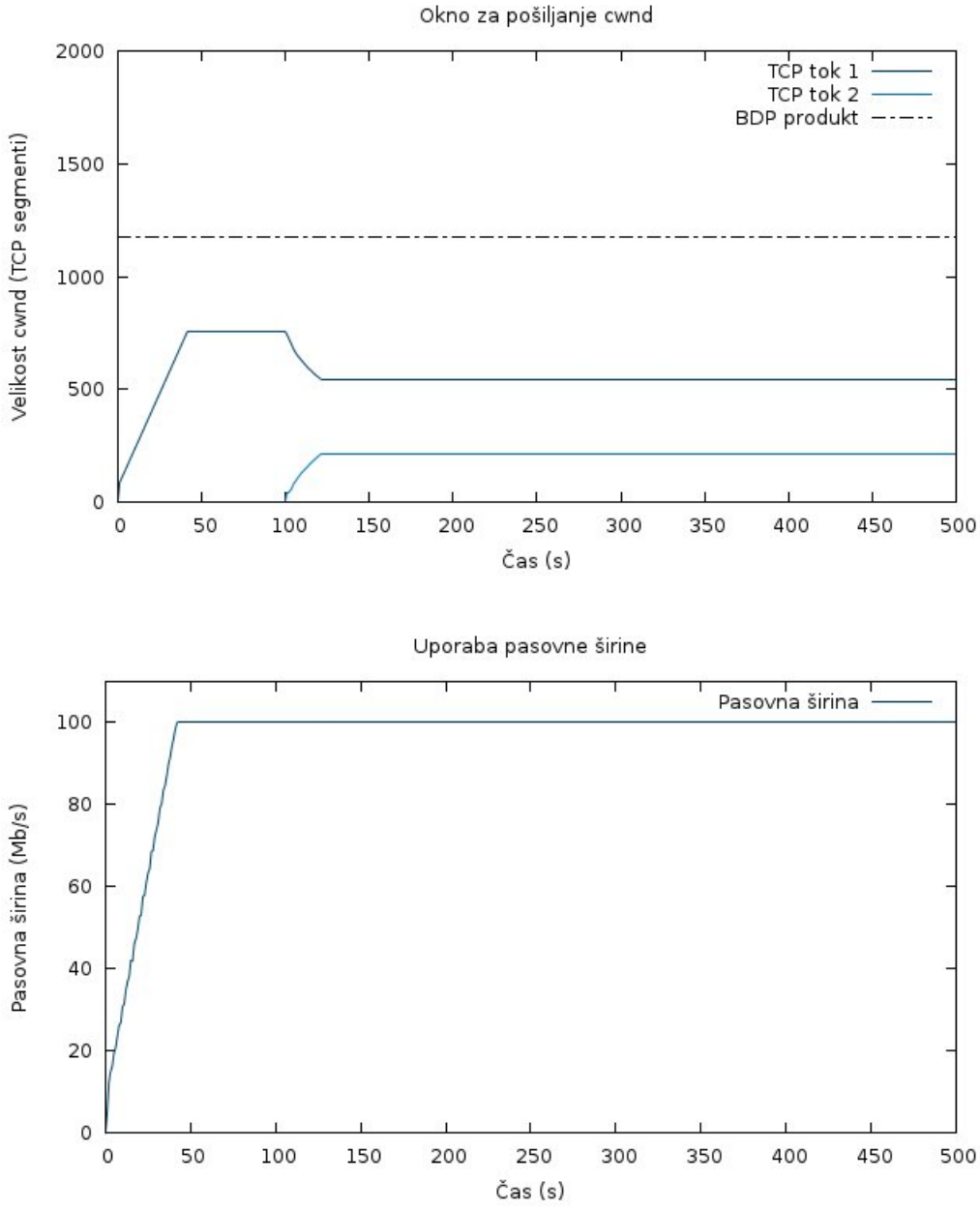
Pri počasnem začetku Vegas primerja pričakovane in dejanske hitrosti pošiljanja. Kadar dejanska hitrost postane manjša kot pričakovana hitrost, Vegas spremeni delovanje algoritma iz počasnega začetka v linearno povečevanje ali linearno zmanjševanje okna.

5.3.4 Rezultati simulacije TCP Vegas protokola

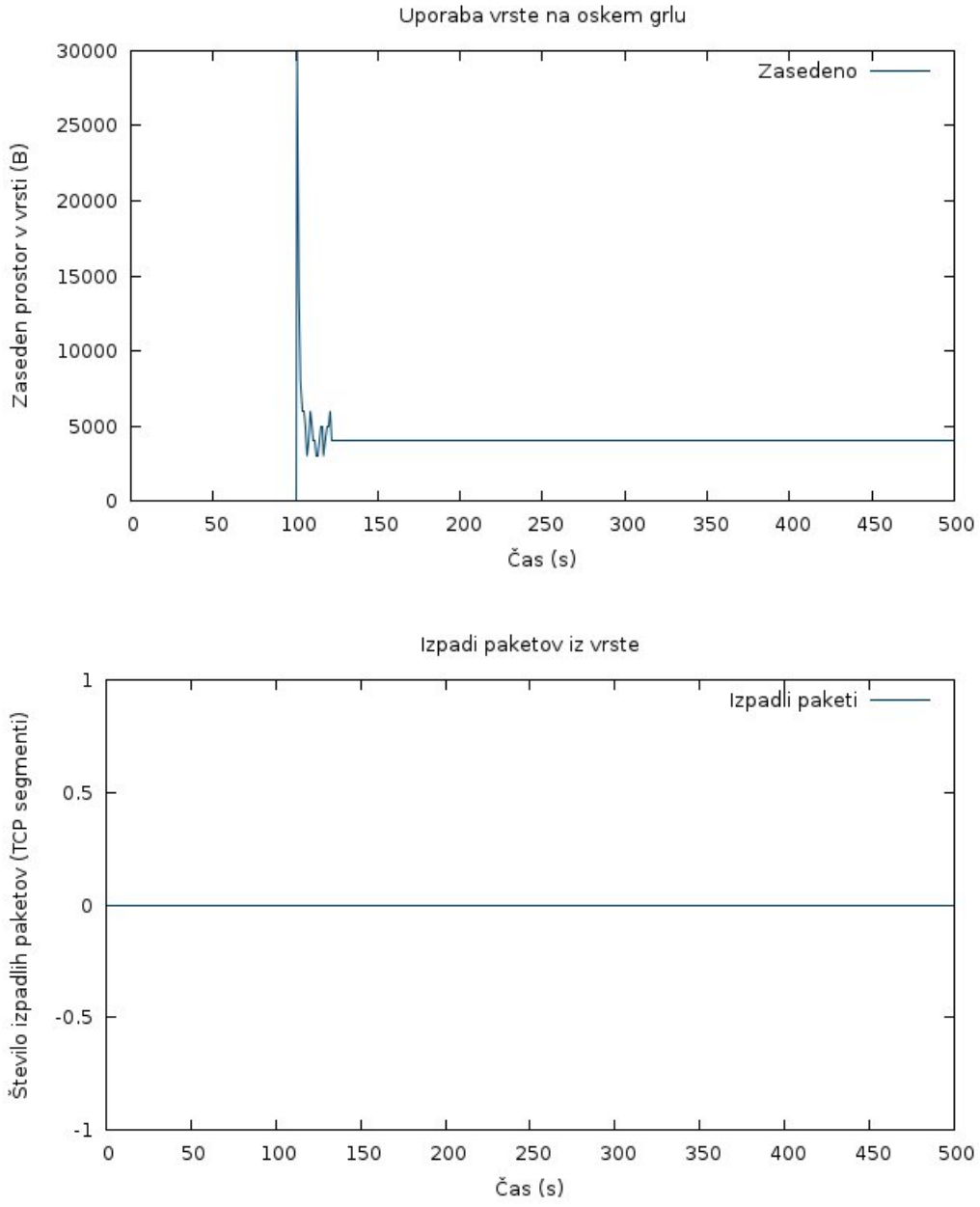
Rezultati simulacije TCP Vegas protokola prikazujejo popolnoma drugačen način izogibanja zamašitvam. Očitno je da algoritem prilagaja hitrost pošiljanja odvisno od prometa. Kadar hitrost pride znotraj določenega predvidenega praga, algoritem enostavno nadaljuje s pošiljanjem z isto hitrostjo. Vrsta je v primeru algoritmov, zasnovanih na zakasnitvah, zelo pomembna. Spremembe v zasedenosti vrste spreminjajo zakasnitve paketov in s tem okno za pošiljanje. Na grafu 5.3.4 se lahko opaža velika sprememba na stoti sekundi, ko se vklopi drugi pošiljatelj. Pri simulaciji ni bilo izgubljenih paketov, kaj pomeni da je TCP Vegas pravočasno prilagajal okna za pošiljanje.

Pomembno je omeniti da pri TCP Vegas protokolu velikost okna spreminja z isto hitrostjo. To pomeni da se okno povečuje več časa na hitrejših povezavah, kar ni dobro za LFN omrežja. Druga pomembna pomanjkljivost je ta, da TCP Vegas ne ujame dovolj pasovne širine v heterogenih omrežjih (tista omrežja po katerih komunicira več različnih podatkovnih tokov, kontroliranih z različnimi algoritmi) [21, 22].

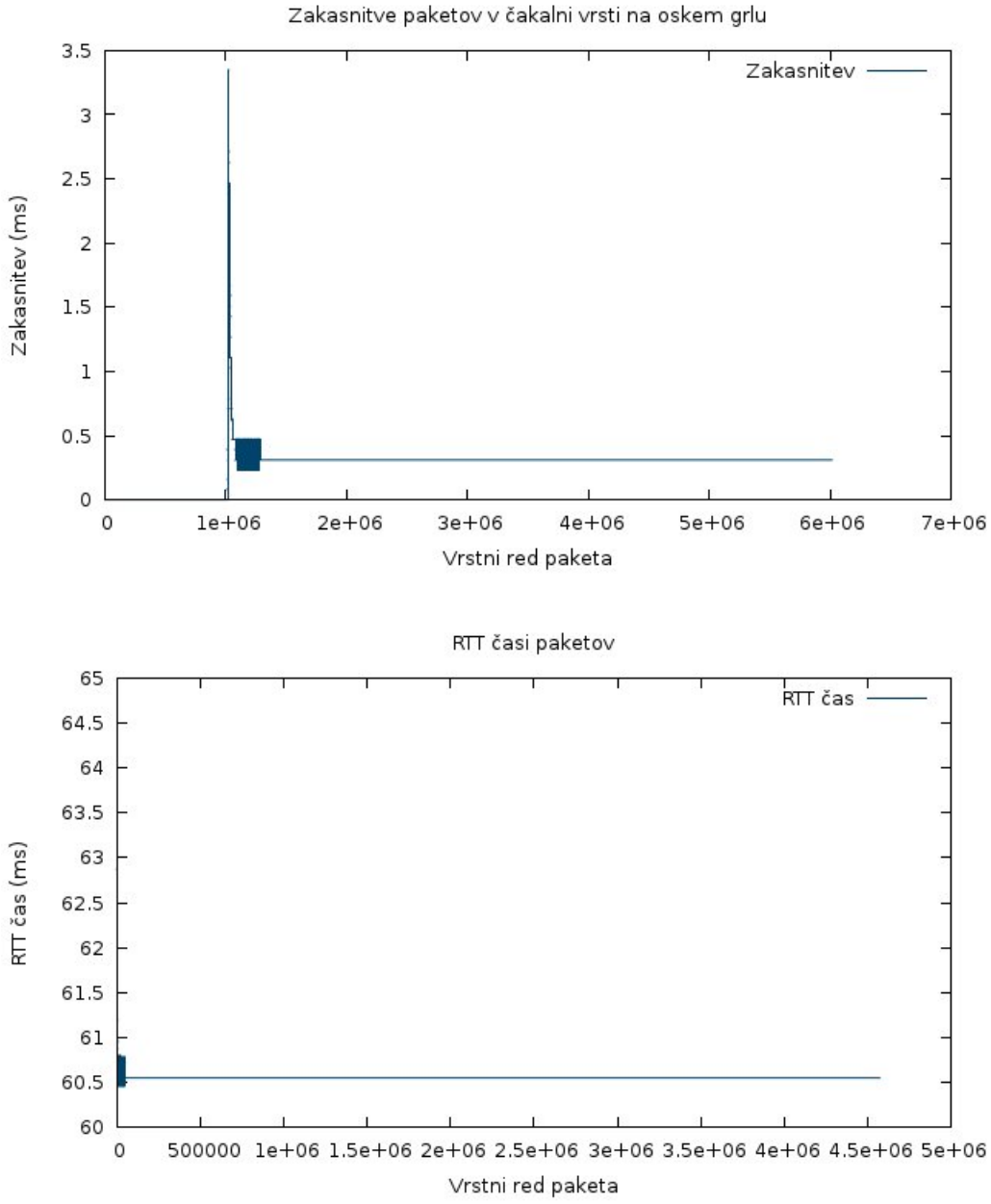
Simulacija je pokazala da ni nujno da dva TCP Vegas toka podelita tok pravilno med sabo. V primeru da oba TCP pošiljatelja začneta pošiljanje istočasno, okno se pravično porazdeli, kot kaže primer 7 v dodatku A. V tem primeru je uporaba vrste močna na samem začetku, potem se ustali.



Slika 5.5: Okno za pošiljanje in pasovna širina TCP Vegas protokola



Slika 5.6: Izkoriščenost in izpadi paketov iz vrste TCP Vegas protokola



Slika 5.7: Zakasnitve v vrsti in RTT čas paketov TCP Vegas protokola

5.4 BIC in CUBIC TCP

BIC TCP [23] je različica TCP protokola, ki je narejena z namenom, da bo čim bolj stabilna. Za nastavitev velikosti okna za pošiljanje uporablja algoritem za binarno preiskovanje. Algoritem uporablja dve velikosti, ki predstavljata minimalno in maksimalno okno. Predpostavlja se da je trenutno dostopna pasovna širina na sredi teh dveh vrednosti. Algoritem hitro povečuje okno kadar je trenutno okno daleč od dostopne kapacitete omrežja (konkavna funkcija). Kadar pride blizu točke nasičenosti, algoritem upočasni povečevanje okna. Primanjkljaj izgub pomeni, da se je trenutno dostopna pasovna širina povečala in TCP tok jo lahko uporabi. V tem primeru se okno začne eksponentno povečevati, da se čim hitreje najde nova maksimalna dostopna pasovna širina (konveksna funkcija).

CUBIC TCP [23] je izboljšal svojega predhodnika z uporabo enostavnejšega algoritma povečevanja okna za pošiljanje. Binarna preiskava se je zamenjala z kubično funkcijo. Takšna funkcija že vsebuje konkavni in konveksni del. Uporaba kubičnih funkcij za določanje velikosti okna pomeni, da je povečevanje okna neodvisno od RTT časa. To omogoča TCP CUBIC podatkovnim tokom, da imajo isto velikost okna pri tekmovanju za trenutno dostopno pasovno širino.

5.4.1 BIC TCP algoritem

Kadar pride do izgub paketov, algoritem zmanjša okno za pošiljanje za faktor β. Potem se vrednosti W_{max} in W_{min} postavita na velikost okna pred izpadom oziroma velikost okna po izpadu. Potem BIC TCP začne binarno preiskavo do srednje točke med tem dvema dejavnikoma. Rast okna čez maksimum pomeni da se okno lahko še poveča. V tem primeru se išče novi W_{max}. Na začetku okno raste počasi, potem začne hitro linearno naraščati.

5.4.2 CUBIC TCP algoritem

CUBIC TCP poenostavlja funkcijo binarnega preiskovanja z uporabo kubične funkcije. Na ta način se ohrani stabilnost algoritma, poveča se pa prijaznost do ostalih TCP tokov pri dodeljevanju pasovne širine.

CUBIC TCP se pri izgubah obnaša podobno BIC TCP-u. Okno za pošiljanje se zmanjša za faktor β, ki znaša 0.2. W_{max} se nastavi na velikost okna pred izgubo. Potem se kubična funkcija nastavi na način, da je meja med konkavnim in konveksnim delom točno na točki W_{max}. Na ta način se zmanjša število

izpadlih paketov v primeru, da ni prišlo do sprememb v uporabni pasovni širini.

$$w(t) = C(t - K)^3 + W_{max} \tag{5.4}$$

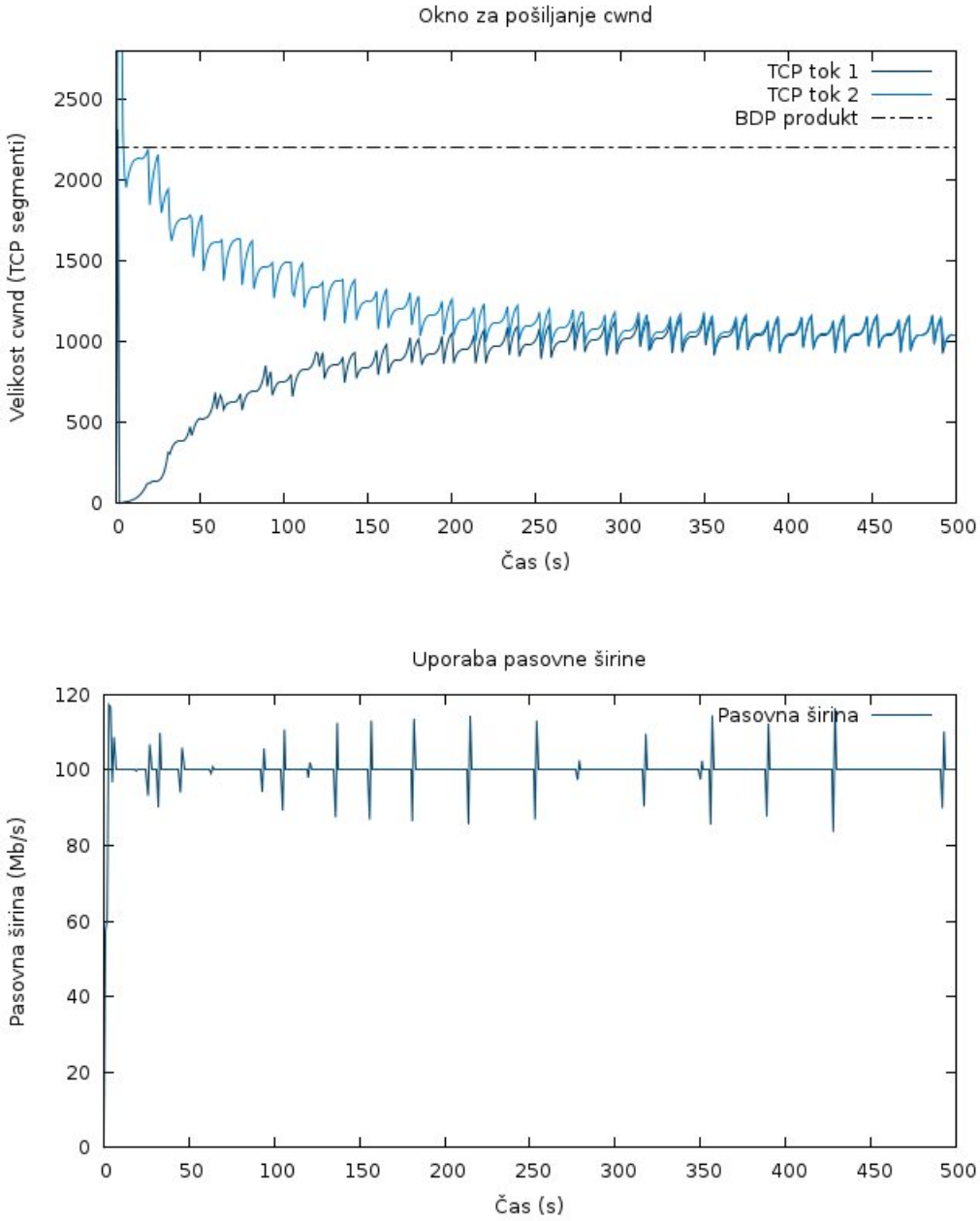
Okno za pošiljanje se računa po funkciji 5.4, kjer C določa hitrost konvergence, oziroma kako hitro funkcija pride do točke spremembe iz konkavnega dela v konveksni in potem kako hitro narašča v konveksnem delu. K je čas, katerega funkcija 5.4 rabi, da poveča okno W do velikosti W_{max} če ni novih izgub, t je čas, ki je potekel od zadnjega zmanjšanja okna. K se računa po enačbi 5.5.

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \tag{5.5}$$

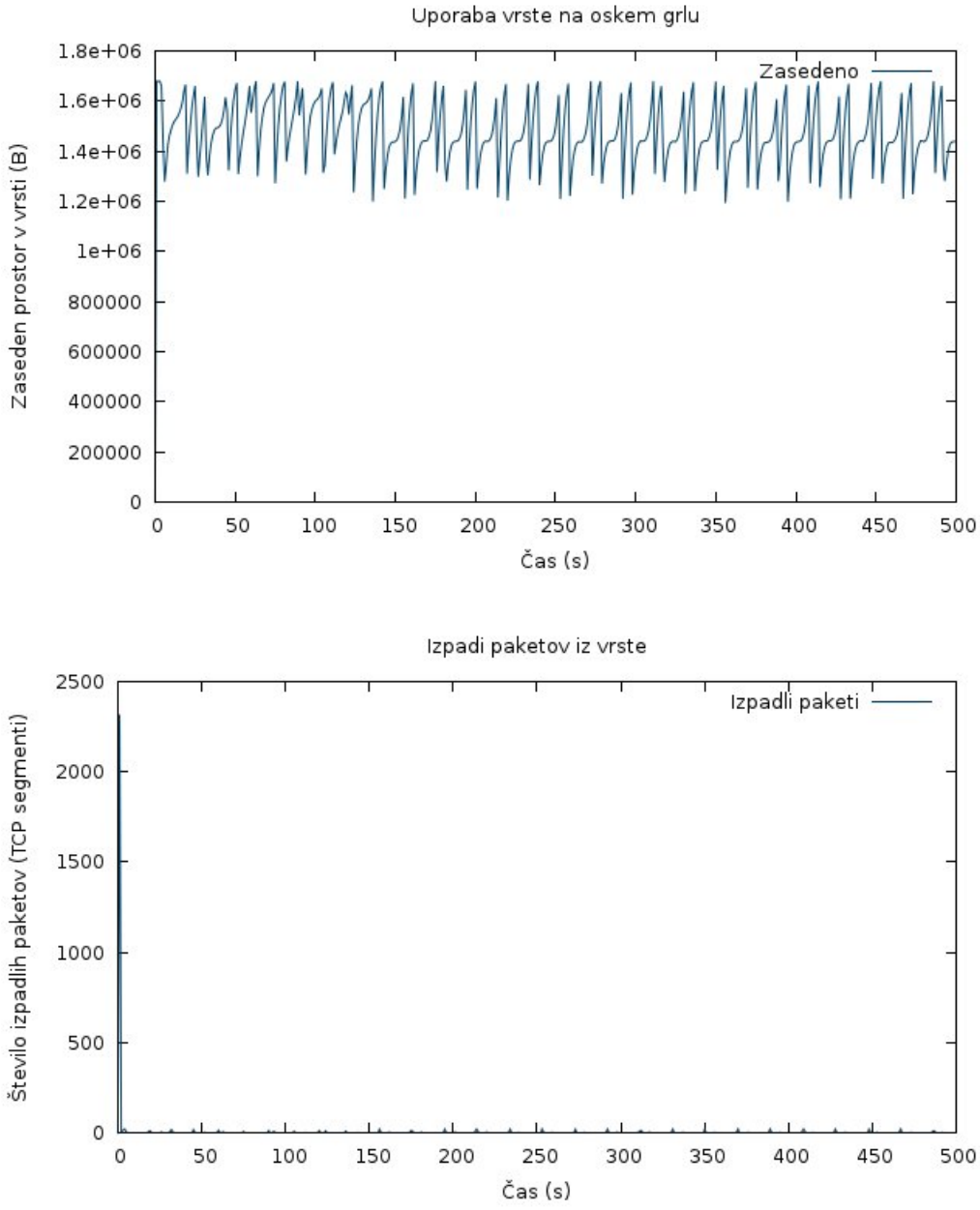
5.4.3 Rezultati simulacije Cubic TCP protokola

Cubic TCP je edini protokol, ki deluje na osnovi izpadlih paketov in ki upočasni pošiljanje preden doseže točko optimalnega pošiljanja. Ostali protokoli v eksponentnem povečevanju pogosto izgubijo večje število paketov, dokler TCP Cubic poskuša zmanjšati to število na minimum z posebno prilagojenim oknom. Paketi rabijo več časa v vrstah (graf 5.4.3), ampak se vrsta ne izprazni pri vsaki izgubi, kot je to slučaj pri TCP Tahoe oziroma TCP Reno. Zaradi tega velikost okna manj niha in se v istem času lahko prenese več podatkov.

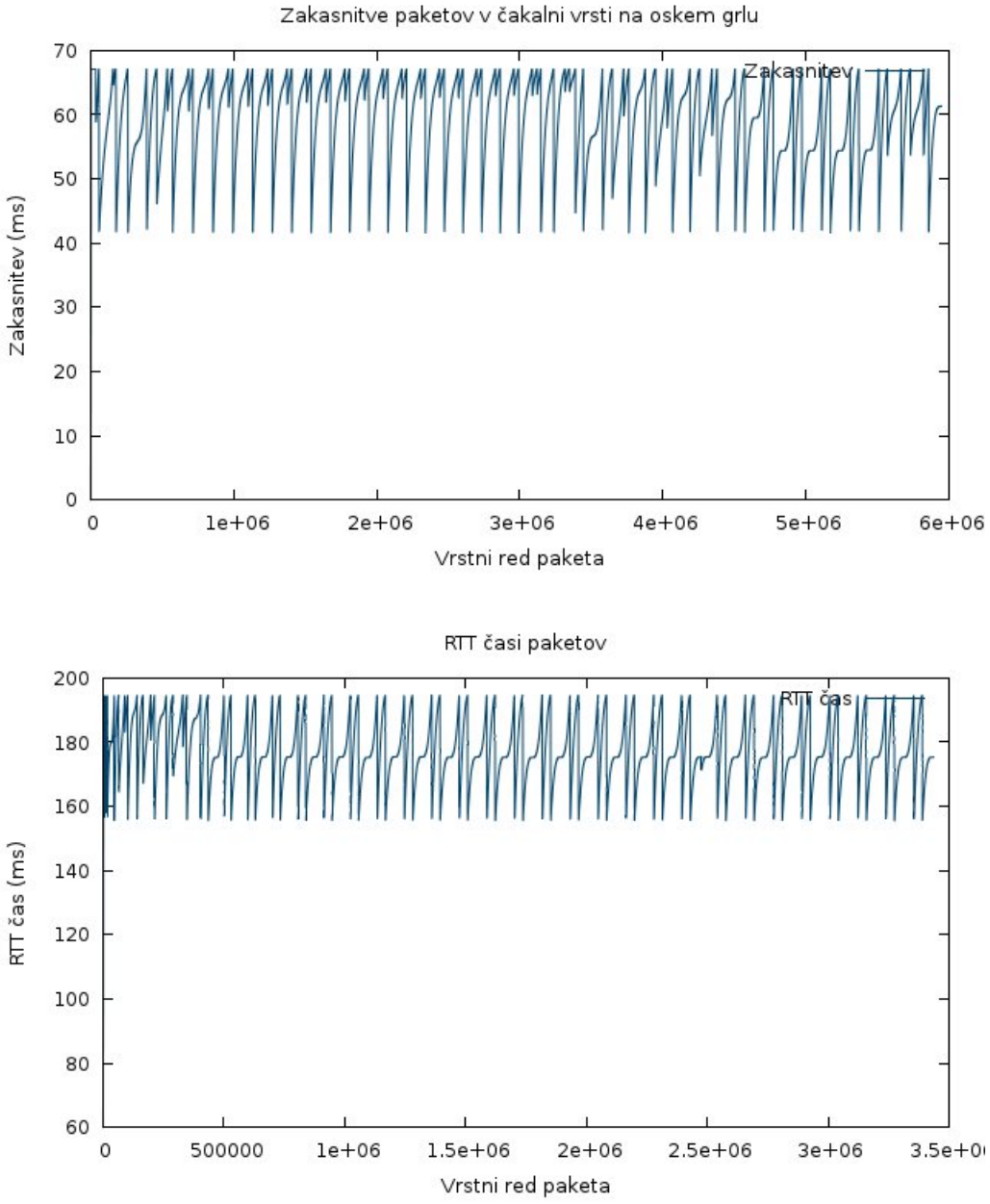
Simulacije so pokazale da se Cubic TCP slabše prilagaja pri manjših vrstah. V prikazani simulaciji je velikost okna zvišana z 800 segmentov na 1600 segmentov. Graf 7 v dodatku prikazuje spreminjanje `cwnd` okna pri uporabi vrste z velikostjo 800 segmentov.



Slika 5.8: Okno za pošiljanje in pasovna širina TCP Cubic protokola



Slika 5.9: Izkoriščenost in izpadi paketov iz vrste TCP Cubic protokola



Slika 5.10: Zakasnitve v vrsti in RTT čas paketov TCP Cubic protokola

5.5 TCP Hybla

Današnja heterogena omrežja so sestavljena od različnih vrst kabelskih, radijskih in satelitskih povezav. Brezžične povezave pogosto imajo veliki čas RTT zaradi velikih razdalj in večjega števila izgub paketov. Večina ostalih algoritmov povečuje svoje okno za pošiljanje odvisno od časa RTT. Ker je verjetnost, da se paket v omrežju uniči zelo majhna, ti algoritmi sklepajo, da izguba paketa pomeni zamašitev omrežja. Po drugi strani, pri brezžičnih povezavah se paketi zlahka uničijo pri prenosu. Zaradi časa potrebnega za ponovno pošiljanje se okno počasi povečuje. TCP Hybla [24] rešuje ta problem tako, da okno povečuje neodvisno od časa RTT.

5.5.1 TCP Hybla algoritem

Osnovna ideja TCP Hybla algoritma je pridobiti hitrost pošiljanja v primeru, da ta TCP povezava ima zelo majhno število napak (torej kakšna bi bila hitrost pošiljanja za isto pasovno širino in čas za širjenje signala, v primeru, da v povezavi ni napak). Za to nalogo Hybla uporablja normalizirani RTT čas ρ, kot je navedeno v enačbi 5.6.

$$\rho = RTT/RTT_0 \tag{5.6}$$

V tej enačbi RTT_0 predstavlja RTT čas povezave, katere zmogljivost se želi doseči. Večina standardnih TCP različic (Tahoe, Reno, New Reno) uporablja enačbo 5.7 [25]. Iz te enačbe je razvidno da se v času počasnega začetka (SS) okno povečuje eksponentno, dokler se v času izogibanja zamašitvam (CA) okno povečuje linearno. Oznaka γ predstavlja prag počasnega začetka *ssthresh*, ki se doseže v času t_γ.

$$W(t) = \begin{cases} 2^{t/RTT}, & 0 \leq t < t_\gamma, \quad \text{SS} \\ \frac{t-t_\gamma}{RTT} + \gamma, & t \geq t_\gamma, \quad \text{CA} \end{cases} \tag{5.7}$$

TCP Hybla spremeni to funkcijo tako, da prilagodi velikost okna in čas s pomočjo normaliziranega časa RTT, definiranega v prejšnji enačbi. Kot je razvidno iz enačbe 5.8, trenutni čas in celotna velikost okna sta zmnoženi z ρ.

$$W^H(t) = \begin{cases} \rho 2^{\rho t/RTT}, & 0 \leq t < t_\gamma, \quad \text{SS} \\ \rho[\rho\frac{t-t_\gamma}{RTT} + \gamma], & t \geq t_\gamma, \quad \text{CA} \end{cases} \tag{5.8}$$

V tem primeru je oznaka t_γ čas, potreben da se doseže prag $\rho\gamma$, in je ista ta vse čase RTT. Iz te enačbe se da dobiti potrebno hitrost za pošiljanje segmentov $B^H = W^H/RTT$, definirano v enačbi 5.9.

$$B^H(t) = \begin{cases} \frac{2^{t/RTT_0}}{RTT_0}, & 0 \leq t < t_\gamma, \quad \text{SS} \\ \frac{1}{RTT_0}[\frac{t-t_\gamma}{RTT_0} + \gamma], & t \geq t_\gamma, \quad \text{CA} \end{cases} \tag{5.9}$$

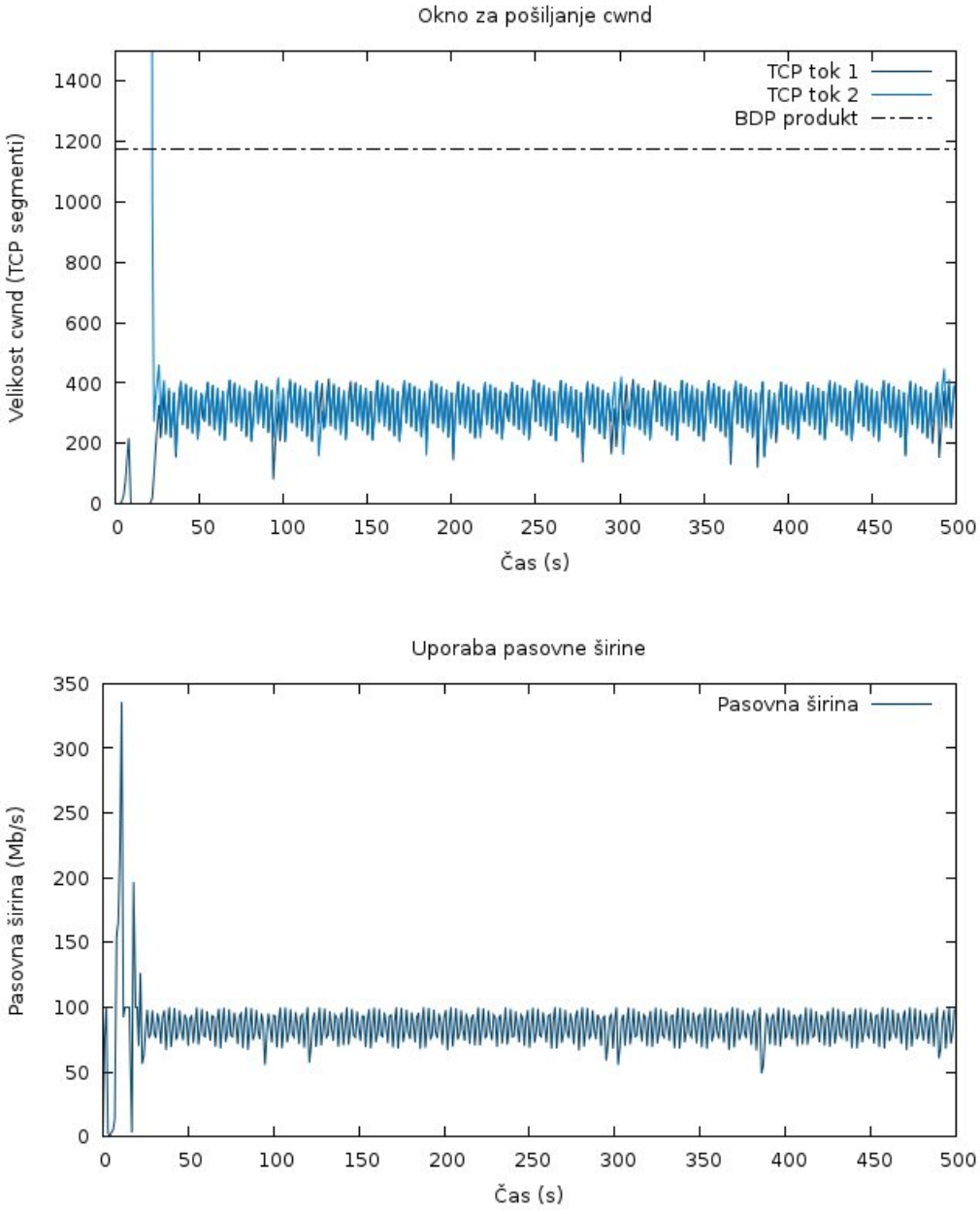
Razvidno je da je hitrost pošiljanja segmentov neodvisna od realnega časa RTT. Namesto tega je enaka hitrosti pošiljanja omrežja z malim številom napak. Za 'hitra' omrežja (omrežja brez velikega števila napak) se TCP Hybla obnaša kot navaden TCP protokol. V tem primeru se ρ namesti na njegovo minimalno vrednost 1. Velikost praga za počasni začetek se lahko izračuna s pomočjo opazovanja zakasnitev med prihodih potrditvenih paketov. Zelo pomembno je določiti točno vrednost za RTT_0. Najboljša izbira je ponavadi RTT povezave, zmanjšan za zakasnitve povzročene zaradi izgub v radijski povezavi.

5.5.2 Rezultati simulacije TCP Hybla protokola

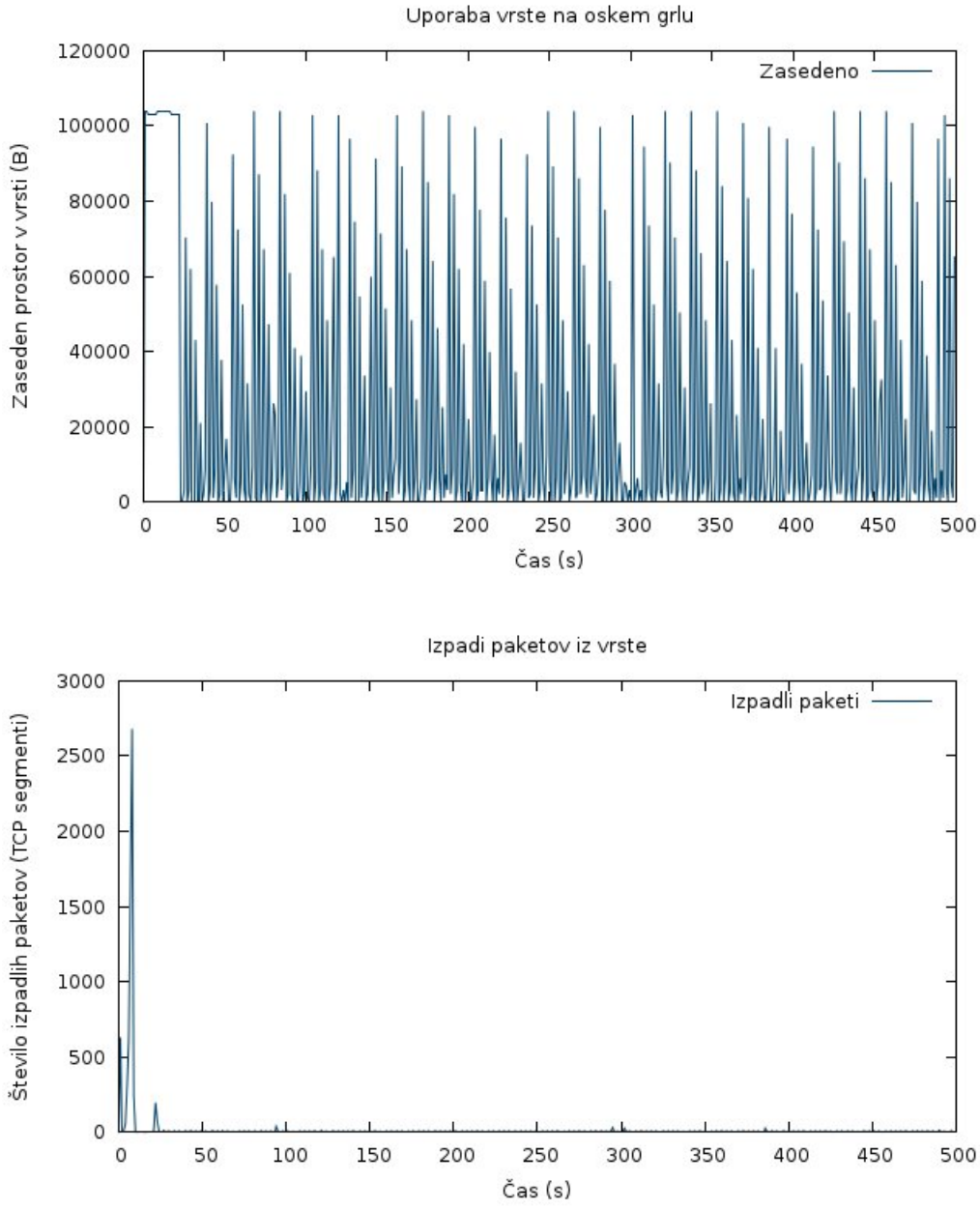
Pri simuliranju algoritma TCP Hybla je velikost vrste zmanjšana na 100 segmentov. Pri samem začetku pošiljanja TCP Hybla poskuša določiti velikost `ssthresh` praga. Algoritem pošlje veliko količino podatkov in z opazovanjem prihajajočih potrditvah določi grobi približek BDP produkta. Potem začne hitro oscilacijo pri robu BDP produkta. Iz grafa izpadlih paketov 5.5.2 je razvidno, da se po vsakem izpadlem paketu okno zmanjša in potem kar hitro spet naraste. Hitre oscilacije okna `cwnd` povzročajo tudi oscilacije na vseh ostalih grafih. Nekatere raziskave trdijo, da velike oscilacije pri TCP Hybla protokolu zmanjšajo stabilnost omrežja [25], kar pomeni več izpadlih paketov. To je razvidno tudi iz tabele 5.1. Na koncu je opazna tudi manjša uporaba pasovne širine (graf 5.5.2).

Pri velikosti vrste 200 segmentov in več se velikost okna `cwnd` stabilizira na določeni velikosti, kot kaže graf 7 v dodatku. V tem primeru je hitrost pošiljanja Hyble tako nastavljena da ni izpadov iz vrste, razen na samem začetku. Grafa zasedenosti vrste 5.5.2 in 7 se očitno razlikujeta v izpadih. V prvem grafu po vsakem izpadu se zasedenost vrste zmanjša, dokler drugi graf prikazuje konstantno uporabljeno vrsto.

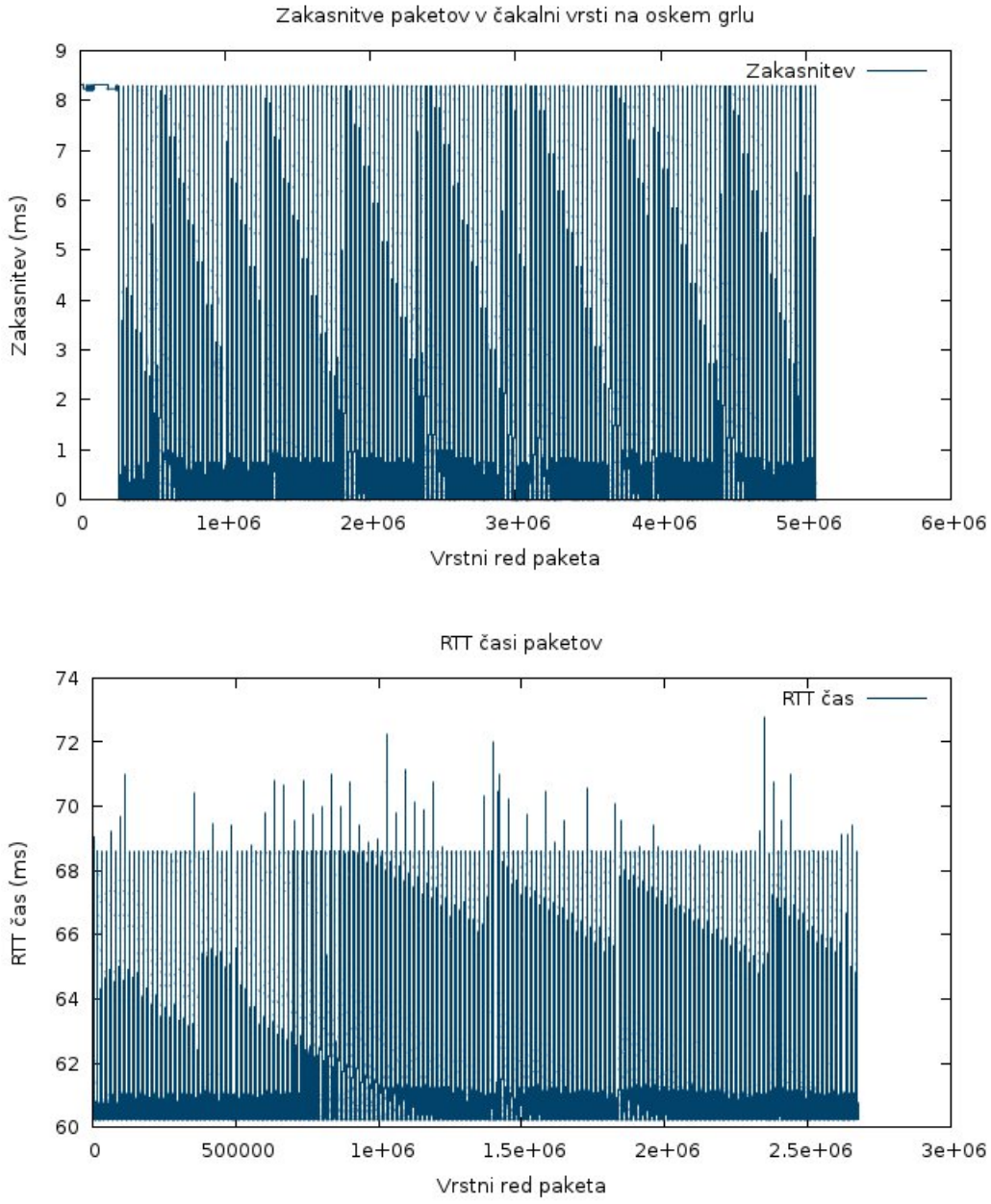
Bolj realna simulacija TCP Hybla protokola bi bila izvedena na modelu podobnem satelitskim povezavam. Takšen model bi imel umetno ustvarjene izgube in večji čas širjenja signala. Poleg tega se lahko moč signala spreminja.



Slika 5.11: Okno za pošiljanje in pasovna širina TCP Hybla protokola



Slika 5.12: Izkoriščenost in izpadi paketov iz vrste TCP Hybla protokola



Slika 5.13: Zakasnitve v vrsti in RTT čas paketov TCP Hybla protokola

5.6 Compound TCP

Compound TCP [26] je algoritem implementiran v Microsoft Windows operacijskih sistemih. Za prilagajanje hitrosti pošiljanja uporablja dvojni pristop opazovanja izgub v omrežju in opazovanja zakasnitvah v vrstah. Na ta način poskuša hitro pridobiti pošteni del pasovne širine za pošiljanje in potem zmanjšati izgube v vrstah na minimum.

5.6.1 Compound TCP algoritem

Algoritem uvaja novo spremenljivko `dwnd`, ki predstavlja okno za pošiljanje, izračunano s pomočjo zakasnitvah v vrstah. Okno za pošiljanje je izračunano z funkcijo 5.10, v kateri `awnd` predstavlja oglašeno okno v TCP zaglavju.

$$win = min(cwnd - dwnd, awnd) \tag{5.10}$$

Compound TCP lahko pošlje *cwnd + dwnd* paketov v enem času RTT, tako da je spremenjena funkcija povečevanja okna (5.11). Velikost *dwnd* je na začetku nameščena na 0.

$$cwnd = cwnd + frac1cwnd + dwnd \tag{5.11}$$

Komponenta z osnovo na na zakasnitvah v vrstah se vklopi kadar algoritem pride v fazo izogibanja zamašitvam. Izpeljana je iz algoritma, ki ga uporablja TCP Vegas (5.12).

$$\begin{aligned} Expected &= win/baseRTT \\ Actual &= win/RTT \\ Diff &= (Expected - Actual) * baseRTT \end{aligned} \tag{5.12}$$

Množenje z *baseRTT* v tretji enačbi določa velikost podatkov v vrsti na "ozkem grlu". Zgodnja zamašitev se opazi če je velikost podatkov v vrsti večja kot prag γ. Če je $Diff < \gamma$, povezava ima slabi promet, v nasprotnem primeru je prometna in je potrebno upočasniti pošiljanje. Nastavitev γ je zelo pomembna, kajti prevelika vrednost bo zasedala vrsto, premajhna vrednost lahko povzroči napačen signal za zgodnjo zamašitev. Nastavljena je na 30 paketov [26].

Kadar ni zamašitev v povezavi, okno se povečuje po enačbi 5.13, kadar zamašitev obstaja, okno se multiplikativno zmanjšuje po enačbi 5.14. Parametri α, β in k so nastavljivi da se lahko dosežejo zaželena skalabilnost in odzivnost, nastavljeni so na vrednosti 1/8, 1/2 oziroma 0.75 [26].

$$win(t+1) = win(t) + \alpha * win(t)^k \tag{5.13}$$

$$win(t+1) = win(t) * (1 - \beta) \tag{5.14}$$

5.6.2 Rezultati simulacije TCP Compound protokola

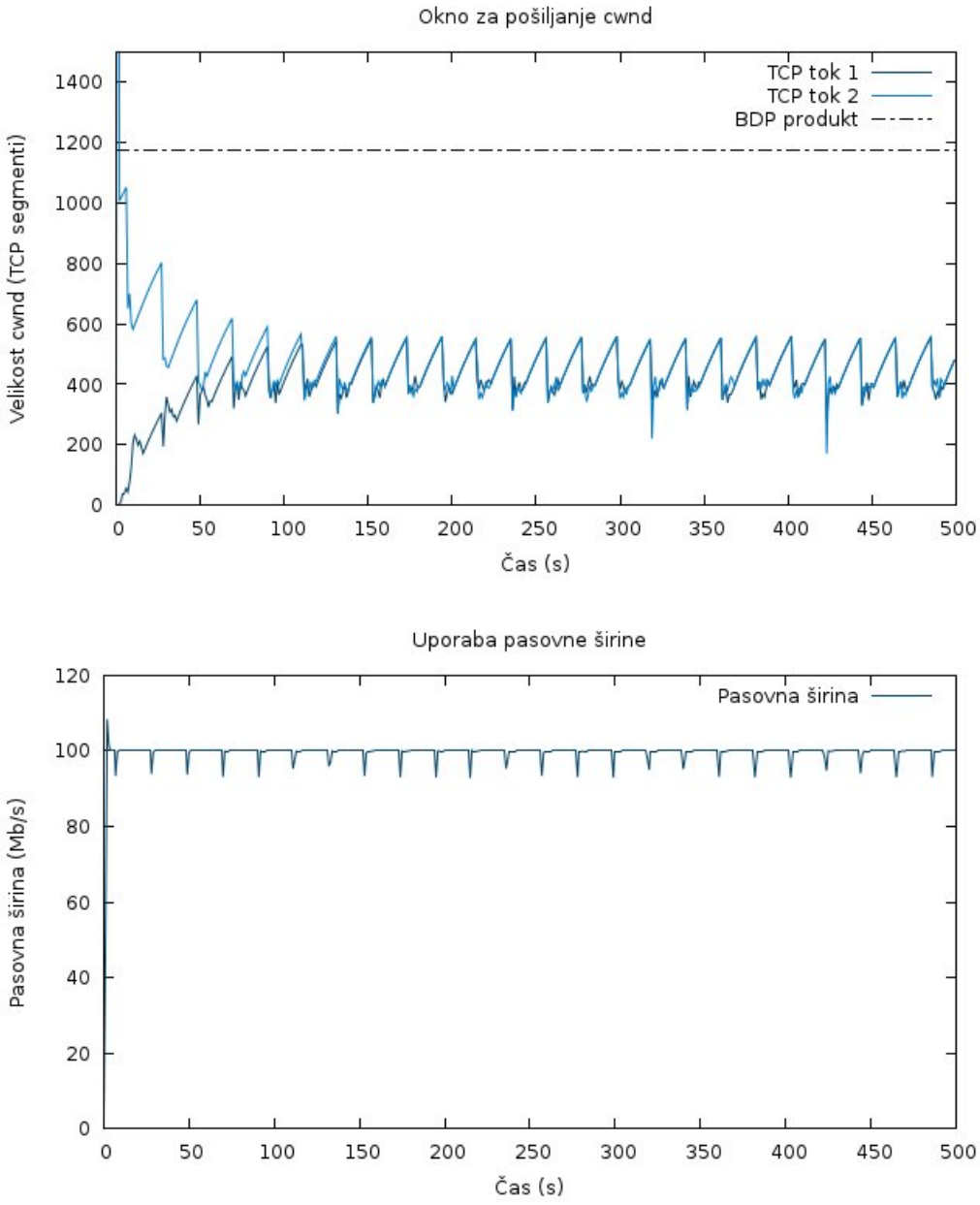
Kot je že povedano, Compound TCP vzdržuje svoje okno za pošiljanje s pomočjo dveh komponent, `cwnd` in `dwnd`. Algoritem torej uporablja merjenje zakasnitev kot tudi izpade paketov za določanje velikosti okna. Zaradi tega je okno sestavljeno iz dveh komponent.

Vrsta na "ozkem grlu" pri tej simulaciji lahko prejeme 400 paketov. Na začetku okno niha okoli določene vrednosti. Potem začne naraščati do izpada paketa. Po izpadu paketa se okno spet zmanjša da ne pride do zamašitve. Na ta način algoritem lahko bolj uspešno tekmuje za pasovno širino kot ostali algoritmi, ki delujejo samo na osnovi merjenja zakasnitev paketov v vrstah (Kot primer se lahko vzame TCP Vegas). Uporaba vrste (graf 5.6.2) narašča v delu, kjer se okno povečuje do izpada paketa, ampak večina paketov v vrsti ima manjše zakasnitve, katere vplivajo tudi na RTT čas (graf 5.6.2.

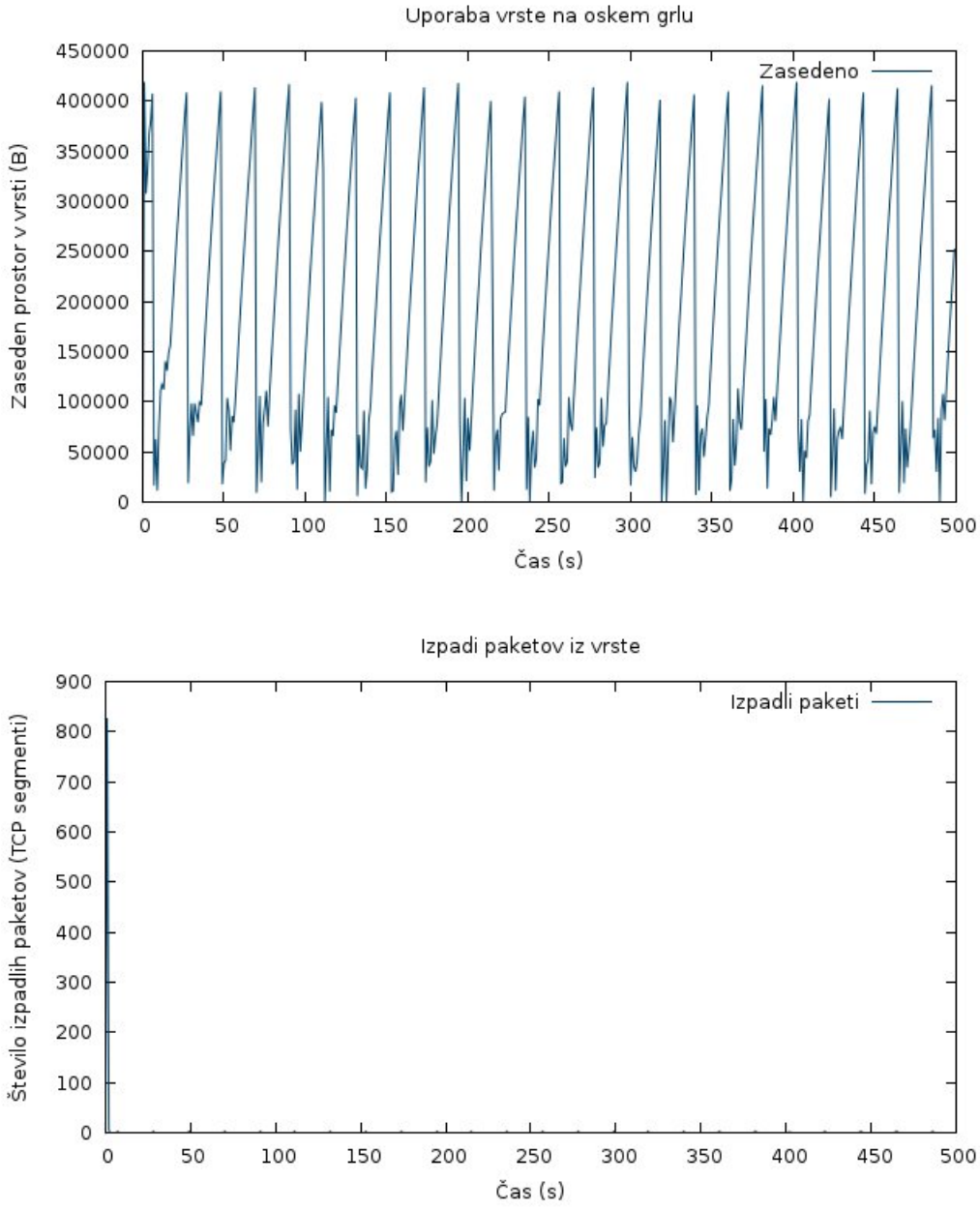
Pri uporabi večje vrste, primeroma velikosti 800 paketov, algoritem ne upočasnjuje pošiljanje, temveč se takoj začne faza povečevanja okna *additive increase*. Če je razlika $Diff$ manjša kot določa prag γ, zgodnja zamašitev ne obstaja in ni treba upočasniti pošiljanje. Takšno stanje prikazuje graf 7 v dodatku. Na grafu sprememb `cwnd` okna je razvidno da variacije okna pred rastom ne obstajajo.

	TCP Reno	TCP Vegas	TCP Cubic	TCP Hybla	TCP Compound
Poslani paketi	5930921	6153602	5950160	5949119	5948245
Sprejeti paketi	5929360	6153602	5948276	5943633	5946661
Izgubljeni paketi	1561	0	1884	5486	1584
Povprečna zasedenost vrste	472540.64	3864.0	741451.2	752398.5	493707.9
Povprečni čakalni čas v vrsti	37.789570339	0.383708382637	59.3937379215	60.3547491786	39.548990409
Povprečni RTT čas	97.8391894665	60.6231315715	119.465758265	120.988123257	99.5072454623

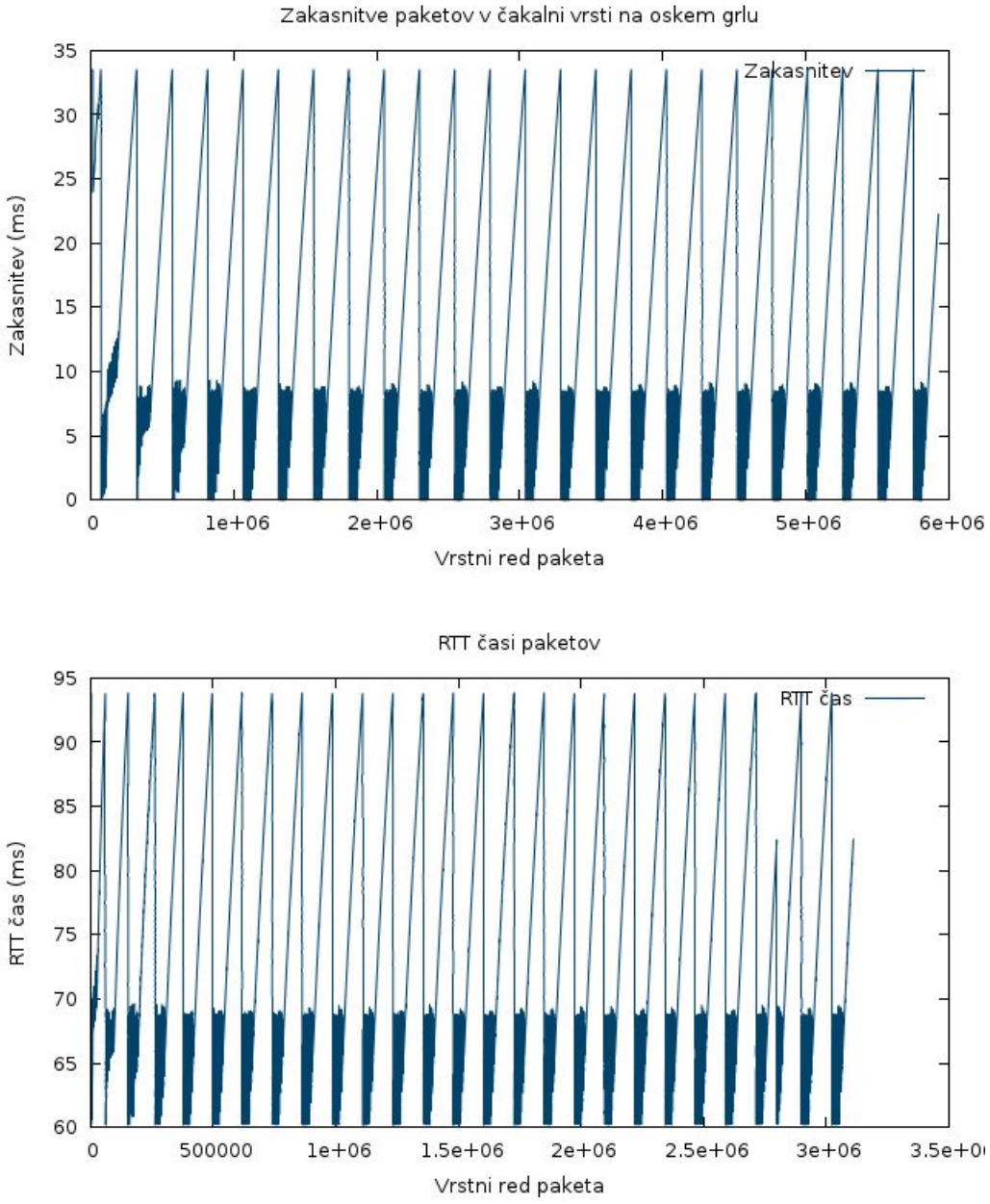
Tabela 5.1: Povprečni rezultati simulacij TCP protokolov. Vsi rezultati so dobiti iz modela, opisanega na sliki 5.1. Vrsta lahko sprejeme 800 paketov.



Slika 5.14: Okno za pošiljanje in pasovna širina TCP Compound protokola



Slika 5.15: Izkoriščenost in izpadi paketov iz vrste TCP Compound protokola



Slika 5.16: Zakasnitve v vrsti in RTT čas paketov TCP Compound protokola

Poglavje 6

Zaključek

Glavni cilj diplomske naloge je bilo raziskati in simulirati različne TCP algoritme za preprečevanje zamašitev na današnjih omrežjih. Poskus simulacije na najnovejših omrežjih z pasovno širino 40 Gbps in 100 Gbps ni uspel. Izkazalo se je da takšne simulacije zasedajo veliko diskovnega prostora in RAM spomina, ter jih ni bilo mogoče izvesti na uporabljenem računalniku. Namesto tega so predstavljeni počasnejši modeli in z nekaj dodatnih simulacij so predlagane morebitne spremembe protokolov na hitrejših modelih.

Zapažene težave se večinoma nanašajo na velikost spomina, potrebnega za popolno uporabo tako velikih omrežjih in trenutno največjo možno velikost `cwnd` okna. Največje oglašeno TCP okno iznaša 65535 B, skupaj z uporabo TCP možnosti za povečevanje okna iznaša 1,073,725,440 B. Omrežje z hitrostjo 100 Gb/s in latenco v eno stran 100 ms ima produkt BDP 1.25 GB, kar je večje kot danes največje možno okno za pošiljanje. Pri takšnih velikih oknih je problem tudi dovolj veliki spominski prostor, rezerviran za pošiljanje in sprejem. Oglaševanje velikega sprejemnega okna pomeni tudi dovolj rezerviranega spomina za sprejem. Iz tega izhaja da bodo v bližji prihodnosti takšna omrežja uporabljana za prenos večjega števila podatkovnih tokov hkrati in ne da bi se uporabljala za hitrejše prenose malega števila podatkovnih tokov.

Simulacije so pokazale da TCP Vegas uspešno pošlje in sprejeme največje število paketov, če ni nobenih drugih aktivnih vrst podatkovnih tokov. Kot je že povedano, Vegas ima slabo stran pri tekmovanju za pasovno širino z drugimi vrstami TCP protokolov [21, 22]. Iz tabele 5.1 je razvidno, da večja uporaba vrste pomeni daljše čakalne čase v vrstah na usmerjevalnikih in večje RTT čase paketov. Iz tega izhaja sklep, da se velikost vrst na usmerjevalnikih mora natančno določiti za čim boljši prenos podatkov. Zasedenost velikih čakalnih vrst lahko pomeni veliki RTT čas. Način spreminjanja in velikost `cwnd` okna sta

pomembna dejavnika pri uporabljeni pasovni širini. Algoritmi z večjim povprečnim oknom za pošiljanje lahko pošljejo več paketov v določenem časovnem intervalu. Zaradi tega se TCP Reno ne prilagaja dobro velikim hitrostim. Razvidno je da TCP Reno pošlje manjše število paketov kot ostali TCP protokoli, ki imajo zmanjševanje okna bolj prilagojeno hitrejšim omrežjem. Okna cwnd na grafih predstavljajo unikatno delovanje vsakega protokola posebej. Spremembe velikosti oken vplivajo na uporabljeno pasovno širino in na zasedenost vrste.

Vsi algoritmi poskušajo čim boljše uporabiti pasovno širino. Zaradi tega so grafi uporabe pasovne širine zmeraj blizu meje 100 Mb/s. Algoritmi, kateri uporabljajo izpade paketov kot informacijo da se je zgodila zamašitev pogosto polnijo vrsto do izpada paketov. Graf 5.2.1 prikazuje uporabo vrste pri TCP Reno. Razvidno je da TCP Reno bolj počasi polni vrsto, ker so konice na grafu bolj narazen. Po drugi strani graf 5.5.2 od TCP Hybla protokola kaže zelo hitro polnjenje vrste. Algoritmi, ki so orientirani na zakasnitve poskušajo izgubiti čim manj paketov. Zakasnitve paketov v čakalni vrsti so zelo odvisne od števila paketov v čakalni vrsti. Algoritmi, ki držijo čakalne vrste zmeraj polne (kot je TCP Cubic) imajo večje zakasnitve ter večji čas RTT. Paket mora čakati da se vsi paketi pred njim pošljejo naprej preden pride na vrsto. Če je v čakalni vrsti malo paketov, paket bo prišel hitro na vrsto, v nasprotnem primeru bo čakanje daljše. Zaradi tega so grafi uporabe vrste in grafi čakanja v vrsti zelo podobni. RTT čas je seštevek več različnih časov. Glavna časa sta čas, potreben za prenos po žici in čas za čakanje v vrsti. Ker je čas za prenos ponavadi zelo podoben za vse pakete (v uporabljenem modelu obstajajo tri povezave od pošiljatelja do sprejemnika z časom širjenja signala 10 ms, torej je skupni čas za prenos v obe smeri približno 60 ms), čas, potreben za čakanje v vrsti da končno obliko grafa RTT časov. Vse simulacije so narejene na diskretnem simulatorju NS2. Ker NS2 deluje na principu dogodkov, ne da bi pošiljal pravih paketov, ni nujno da prikazuje popolnoma točno delovanje algoritmov v realnem omrežju. Zaradi tega so potrebne dodatne raziskave na emulatorju in na realnih omrežjih da se potrdijo dobiti rezultati.

Različni TCP protokoli so narejeni za različne namene. TCP Hybla deluje zelo dobro pri satelitskih povezavah. TCP Compound in TCP Cubic sta uporabna na fiksnih omrežjih. TCP Cubic se lahko uporabi tudi na satelitskih povezavah. Obstaja še veliko različnih implementiranih in predlaganih različic TCP protokola, ki delujejo na različne načine. V današnjih časih večina osebnih računalnikov uporablja novejše algoritme, ki so prilagojeni hitrim LFN omrežjem. Administratorji strežnikov in komunikacijskih centrov z zelo hitrim povezavam rabijo nameniti posebno pozornost pri izbiri in nastavitvah algo-

ritmov za pošiljanje podatkov. Seveda je potrebno vedeti topologijo omrežja ter načine prenosa (brezžični ali fiksni). Pravilno nastavljeno omrežje deluje boljše in hitrejše kot omrežje z tovarniškimi nastavitvami. Uporaba protokolov za hitri prenos ni omejena samo na infrastrukturo Interneta in njegovo hrbtenico, ona pomaga pri prenosu velikega števila podatkov katerekoli vrste. Uporabljajo ih lahko podjetja, ki se ukvarjajo z računalništvom v oblaku (angl. *cloud computing*) za prenos virtualnih strojev. Algoritmi za hitra omrežja so pomembni tudi pri različnih vrstah sinhronizacije, kot so podatkovne baze ali sistemi za prenos vsebine (angl. *content delivery systems*). Na koncu lahko navedemo računalniške omrežne igrice, narejene za veliko število ljudi (več tisoč ljudi lahko igra hkrati na istem strežniku).

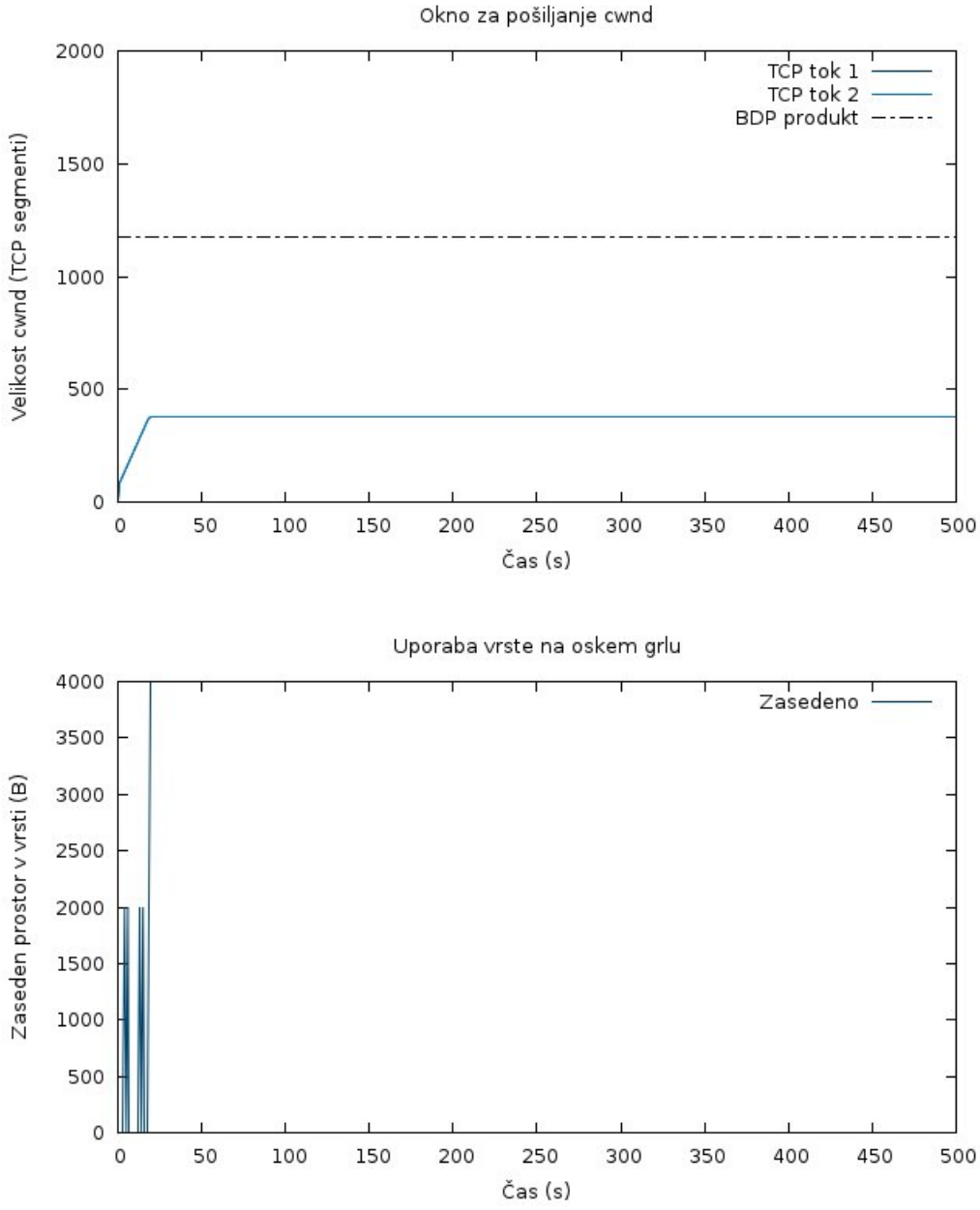
Ker je Internet narejen kot eno veliko heterogeno omrežje, zelo težko je predlagati univerzalno rešitev za najhitrejše pošiljanje. Pri raziskovanju smo prišli do mnenja da se bo kontrola zamašitev prenesla v veliki meri na usmerjevalnike, kjer zamašitve pravzaprav nastajajo. Že obstajajo različne rešitve in algoritmi za omejevanje hitrosti na usmerjevalnikih in sporočanje o zamašitvah pošiljateljem, kot je *Explicit Congestion Notification* (skrajšano *ECN*). Če bi se ti že obstoječi algoritmi izboljšali tako, da pošiljajo popolno stanje pošiljateljem, velikost okna za pošiljanje bi bila regulirana zelo natančno in ne bi jo bilo treba predvidevati ali računati. V tem primeru bi se zmanjšalo tudi število napak, katere se zgodijo pri izpadih paketov iz čakalnih vrst. To bi bila zelo dobra osnova za izdelavo novih protokolov, ki so bolj usmerjeni na čim hitrejše pošiljanje podatkov. Seveda, za takšen projekt bo rabilo veliko časa in ostalih resursov.

Za prihodnja raziskovanja je predlagana emulacija protokolov s pomočjo orodjih, kot je *Dummynet*. Tako bo mogoča simulacija več TCP tokov z različnimi algoritmi za preprečevanje zamašitev, skupaj z UDP tokovi in napakami v omrežju. Na ta način bo pridobit boljši vpogled v prijaznost različnih vrst TCP tokov do porazdelitve pasovne širine med sabo. Za satelitske povezave se lahko umetno naredijo različno porazdeljene napake in različne zakasnitve in moči signalov. Na usmerjevalnikih se lahko namesti podpora za ECN, in na ta način se raziskava preusmeri na preprečevanje zamašitev na usmerjevalnikih.

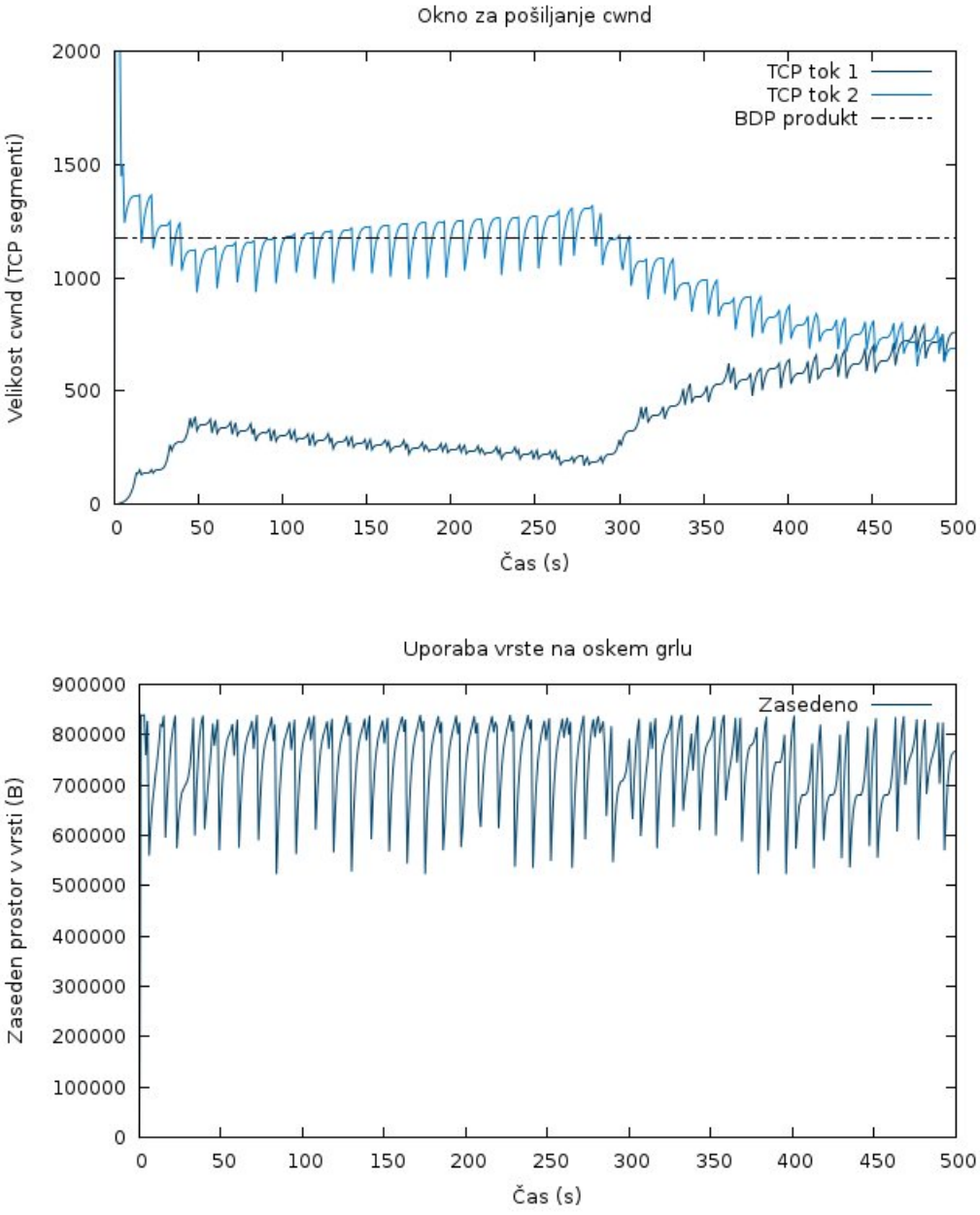
Poglavje 7

Dodatek

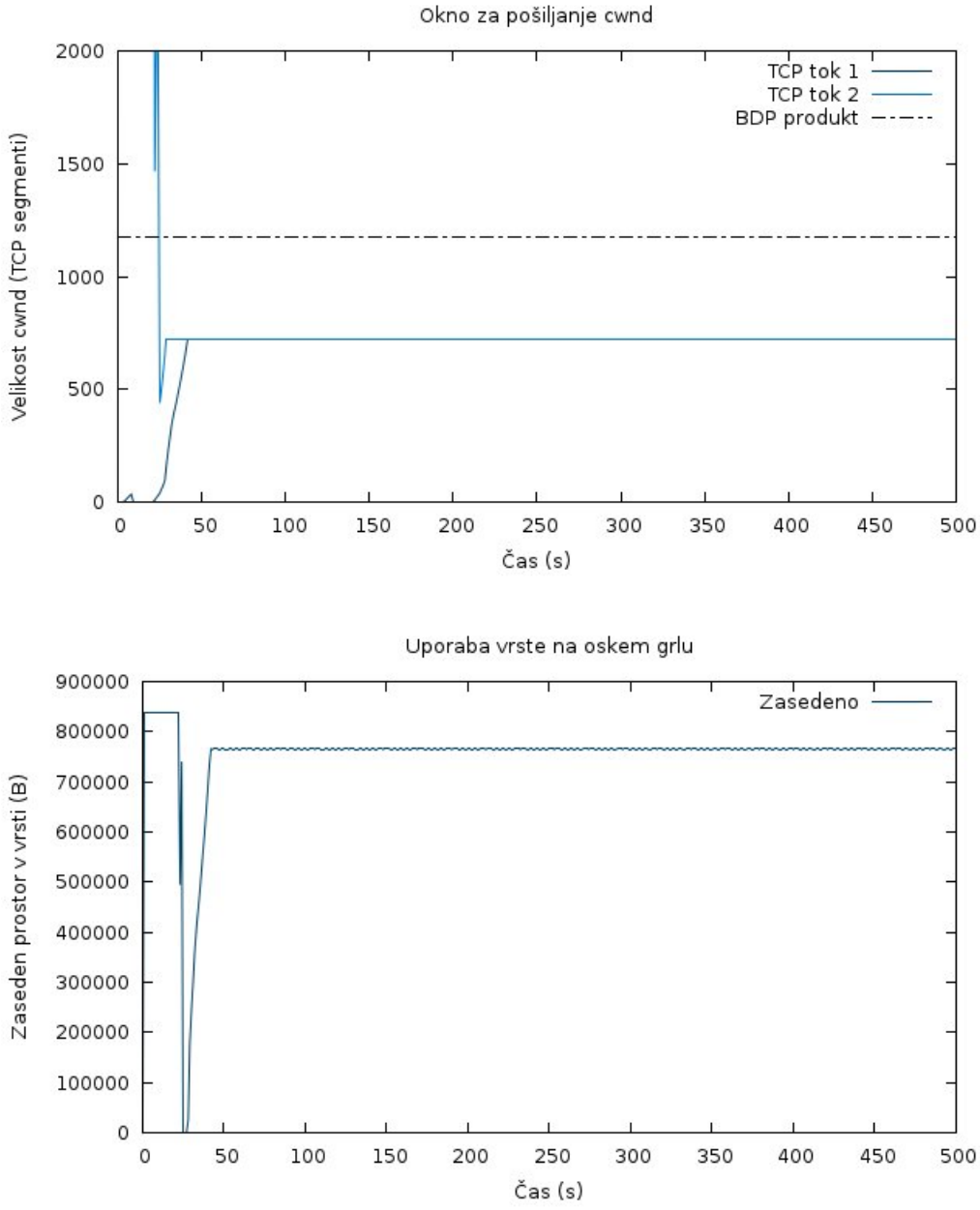
Dodatek vsebuje rezultate dodatnih simulacij protokolov Vegas, Cubic, Hybla in Compound. Za vsak algoritem sta narejena dva dodatna grafa. Prvi prikazuje okno `cwnd` in drugi prikazuje zasedenost vrste.



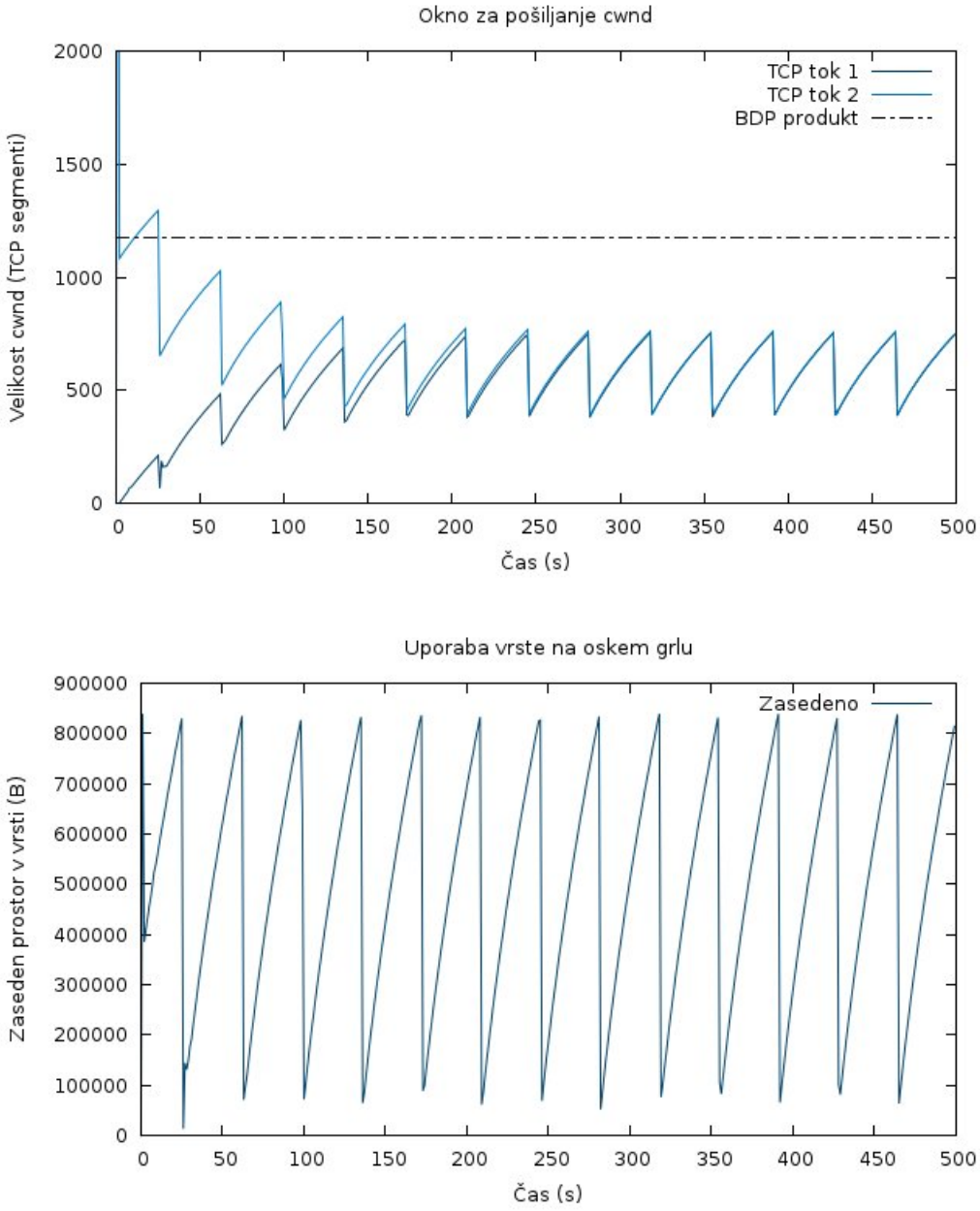
Slika 7.1: Dodatek: Okno za pošiljanje in uporaba vrste TCP Vegas protokola



Slika 7.2: Dodatek: Okno za pošiljanje in uporaba vrste TCP Cubic protokola



Slika 7.3: Dodatek: Okno za pošiljanje in uporaba vrste TCP Hybla protokola



Slika 7.4: Dodatek: Okno za pošiljanje in uporaba vrste TCP Compound protokola

Slike

Tabele

Literatura

[1] Van Jacobson, Michael J. Karels, "Congestion Avoidance and Control", *ACM SIGCOMM Computer Communication Review*, št. 4, zv. 18, str. 314-329, nov. 1988

[2] W. Richard Stevens, *TCP/IP Illustrated, Volume 1: The Protocols, first edition*, Addison-Wesley, Massachusetts, dec. 15, 1993

[3] Information Sciences Institute, *RFC 793: Transmission Control Protocol*, http://tools.ietf.org/html/rfc793, sept. 1981

[4] Information Sciences Institute, *RFC 791: Internet Protocol*, http://tools.ietf.org/html/rfc791, sept. 1981

[5] W. Goralski, *The illustrated network: how TCP/IP works in a modern network*, Morgan Kaufmann Publishers, Massachusetts, dec. 9. 2008

[6] F. Gont, S. Bellovin, *RFC 6528: Defending against Sequence Number Attacks*, http://tools.ietf.org/html/rfc6528, feb. 2012

[7] M. Welzl, *Network Congestion Control*, John Wiley & Sons Ltd, West Sussex, England, jul. 2005

[8] M. Allman, V. Paxson, W. Stevens, Network Working Group, *RFC 2581: TCP Congestion Control*, http://tools.ietf.org/html/rfc2581, apr. 1999

[9] Craig Partridge, *Gigabit Networking*, Addison-Wesley, Massachusetts, 1993

[10] IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force, http://www.ieee802.org/3/ba/index.html

[11] V. Jacobson, R. Braden, D. Borman, *RFC 1323: TCP Extensions for High Performance*, http://tools.ietf.org/html/rfc1323, may 1992

[12] P. Karn, C. Partridge, "Improving round-trip time estimates in reliable transport protocols", *ACM SIGCOMM Computer Communication Review*, št. 5, zv. 17, str. 2-7, nov. 1987

[13] Network Simulator 3 (NS3), http://www.nsnam.org/

[14] OMNeT++, http://www.omnetpp.org/

[15] Opnet Modeler, http://www.opnet.com/solutions/network_rd/modeler.html

[16] Tetcos Netsim, http://tetcos.com/netsim_gen.html

[17] Network Simulator 2 (NS2), http://www.isi.edu/nsnam/ns/

[18] Teerawat Issariyakul, Ekram Hossain, *Introduction to Network Simulator NS2*, Springer, 2009

[19] R. E. Shannon, "Introduction to the art and science of simulation", *Proc. of the 30th conference on Winter simulation*, zv. 1, str. 7-14, dec. 1998

[20] Lawrence S. Brakmo, Larry L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", *IEEE Journal on Selected Areas in Communications*, zv. 13, št. 8, str. 1465-1480, okt. 1995

[21] Luigi A. Grieco ,Saverio Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control", *ACM SIGCOMM Computer Communication Review*, zv. 34, št. 2, str. 25-38

[22] Ghassan A. Abed, Mahamod Ismail, Kasmiran Jumari, *Australian Journal of Basic and Applied Sciences*, 2011

[23] Sangtae Ha, Injong Rhee, Lisong Xu, "CUBIC: a new TCP-friendly high-speed TCP variant", *ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel*, zv. 42, št. 5, str. 64-74, jul. 2008

[24] Carlo Caini, Rosario Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks", *International Journal of Satellite Communications and Networking*, zv. 22, št. 5, str. 547-566, sept. 2004

[25] Siddharth Trivedi et al, "Comparative performance evaluation of TCP Hybla and TCP Cubic for satellite communication under low error conditions", *2010 IEEE 4th International Conference on Internet Multimedia Services Architecture and Application(IMSAA)*, str. 1-5, dec. 2010

[26] Kun Tan, Jingmin Song, Qian Zhang, Murari Sridharan, "A Compound TCP Approach for High-speed and Long Distance Networks", *Proceedings 25th Conference on Computer Communications (InfoCom 2006)*, apr. 2006