Mjerenje performansi TCP algoritma za kontrolu zagušenja

Gladović, Mišo

SVEUČILIŠTE U RIJECI

FILOZOFSKI FAKULTET

ODSJEK ZA POLITEHNIKU

Mišo Gladović

MJERENJE PERFORMANSI TCP ALGORITMA ZA KONTROLU ZAGUŠENJA

DIPLOMSKI RAD

Rijeka, rujan 2015.

SVEUČILIŠTE U RIJECI

FILOZOFSKI FAKULTET

ODSJEK ZA POLITEHNIKU

Mišo Gladović

Matični broj: 123

Studij: Dvopredmetni diplomski studij politehnike i informatike

MJERENJE PERFORMANSI TCP ALGORITMA ZA KONTROLU ZAGUŠENJA

DIPLOMSKI RAD

Mentor:

prof. dr. sc. Mario Radovan

Rijeka, rujan 2015.

SVEUČILIŠTE U RIJECI - FILOZOFSKI FAKULTET

ODSJEK ZA POLITEHNIKU

Povjerenstvo za diplomske ispite

Br.: 602-04/08-01/87

Rijeka, 25. 9. 2015.

ZADATAK

za diplomski rad

Pristupnik: Matični broj:

Naziv zadatka:

Sadržaj zadatka:

Zadano: Predati:

Mentor: Predsjednik Povjerenstva:

prof. dr. sc. Mario Radovan

izv. prof. dr. sc. Ana Meštrović

Zadatak preuzeo dana:

(potpis pristupnika)

Dostaviti:

- Predsjednik Povjerenstva
- Mentor
- Djelovođa Povjerenstva
- Evidencija studija
- Pristupnik

Sažetak (Abstract)

Glavni razlog zbog kojeg možemo uspješno koristiti Internet je TCP-ova kontrola zagušenja. Postoje mnoge varijante TCP protokola i među njima su TCP Tahoe, TCP Reno i TCP NewReno. Ove varijante koriste različite načine za rješavanje zagušenja. U ovom radu ćemo opisati kako protokol TCP radi u point-to-point vezama, koji su načini da se predvide zagušenja i oporavi od njih, i konačno, koja varijanta daje najbolje rezultate u zagušenim mrežama. Simulacije su izvedene pomoću simulatora mreža Ns-3.

The main reason we can use Internet successfully is the TCP congestion control. There are many variants of the TCP protocol and among those are TCP Tahoe, TCP Reno and TCP NewReno. These variants use different ways to deal with congestions. In this paper it will be described how TCP protocol works in point-to-point connections, what are the ways to predict and recover from congestions and, finally, what variant is getting the best results in congested networks. The simulations are done using the network simulator Ns-3.

Ključne riječi

TCP protokol, Ns-3, zagušenje računalne mreže

Sadržaj

1. Uvod

Računalne mreže nam omogućavaju komunikaciju, razmjenu ideja, informacija i sadržaja. Utječu na gotovo sva područja rada i postale su neizostavan dio života. U samo nekoliko desetljeća računalne mreže su se razvile u velike i kompleksne sustave sa stotinama milijuna umreženih računala, mrežnih komponenti i komunikacijskih veza. S porastom broja umreženih računala i interakcije među njima porastao je i broj zagušenja podatkovnog prometa. Količina razmjenjenih podataka i vremenski period aktivnosti korisnika su promjenjive, nepredvidljive veličine i izravno utječu na pojavu zagušenja koja znatno smanjuju iskorištenost resursa mreže. Stoga se javlja potreba za stalnim i dinamičkim upravljanjem zagušenjem prometa u cilju optimizacije maksimalne propusnosti mreže pri komunikaciji s kraja na kraj.

U TCP/IP[1] modelu mreže, koji je osnova rada Interneta, zadatak upravljanja zagušenjem obavlja TCP protokol na transportnom sloju. Koliko dobro obavlja taj zadatak možemo provjeriti stvarnom implementacijom računalne mreže ili programskom simulacijom mreže. Simulacija računalne mreže je najčešće korištena metoda ispitivanja ponašanja računalnih mreža u različitim uvjetima jer dopušta istraživanja po relativno niskoj cijeni i potrebnom vremenu. Simulatori omogućuju oblikovanje proizvoljne mreže određivanjem čvorova mreže i komunikacijskih kanala.

Cilj ovog rada je opisati protokol TCP i neke od načina kako upravlja zagušenjem podatkovnog prometa te ispitati kako na promet utječu različiti parametri i stanja mreže uz pomoć simulatora računalne mreže Ns-3.

1 TCP/IP je model strukture računalne mreže predstavljen sa četiri sloja. Ime je dobio po najvažnijim protokolima TCP (engl. *Transmission Control Protocol*) i IP (engl. *Internet Protocol*).

2. Protokol TCP

Računalna mreža sastoji se od čvorova (engl. *nodes*) i od komunikacijskih veza među tim čvorovima. Čvorove dijelimo na nekoliko hijerarhijski ustrojenih podsustava koje zovemo slojevima (engl. *layer*). U TCP/IP modelu računalne mreže, a koji čini platformu na kojoj radi Internet, postoje četiri sloja: fizički sloj, mrežni sloj, transportni sloj i sloj aplikacije. Fizički i mrežni sloj pružaju nepouzdan prijenos paketa[2] podataka jer u slučaju kvara na dijelovima mreže paketi mogu biti izgubljeni, uništeni ili dostavljeni izvan ispravnog poretka. Postavljanjem otkrivanja i otklanjanja pogreški na fizičkom ili mrežnom sloju sustav bi se nepotrebno komplicirao iz razloga što se te funkcije u pravilu obavljaju na krajnjim točkama komunikacije[1]. Zbog toga je pouzdanost prijenosa ostvarena protokolima na transportnom sloju mreže.

Najčešće korišteni protokol koji pruža pouzdan transport paketa podataka s kraja na kraj je TCP (engl. *Transmission Control Protocol*). To je spojno orijentiran protokol[3] koji pouzdanost ostvaruje mehanizmima potvrde i ponovnog slanja (retransmisije) a pri tom ima očuvan redoslijed poslanih i primljenih paketa. Pouzdanost je svojstvo da svi poslani paketi do primatelja dođu u neizmijenjenom obliku i u redoslijedu kojim su poslani, dok samo trajanje prijenosa nije ograničeno. S druge strane, drugi najčešće korišteni protokol transportnog sloja, UDP (engl. *User Data Protocol*), ne vodi računa o pouzdanosti te se koristi za aplikacije gdje je potrebno u što kraćem vremenu prenijeti što je moguće više podataka bez osvrtanja na izgubljene ili pogrešne pakete kao npr. video ili audio sadržaji.

Teorijska podloga za implementaciju protokola TCP je objavljena 1974. godine. Na osnovi predloženog rješenja komunikacije sa komutacijom paketa (engl. *packet switching*), TCP je 1983. godine implementiran u postojeću eksperimentalnu mrežu koja je spajala četiri američka sveučilišta ARPANET[4] (engl. *Advanced Research Projects Agency Network*), gdje je zamijenio dotadašnji protokol NCP (engl. *Network Control Protocol*). Protokol TCP je definiran u RFC[5] 793 [2] i od tada je prošao kroz nekoliko preinaka, ali

2 paket (engl. *packet*) je jedinstvena struktura koja predstavlja jedinicu podataka koja se prenosi kroz sve dijelove računalne mreže.

3 spojna orijentiranost (engl. *connection oriented*) označava vrstu komunikacije gdje se prije početka slanja podataka uspostavlja izravna veza između računala sudionika komunikacije.

4 projekt agencije DARPA (engl. Defense Advanced *Research Projects Agency*).

5 RFC (engl. *Request for Comments*) su dokumenti izdani od strane *Internet Engineering Task Force* (IETF). Dokumenti obično opisuju metode, ponašanja, istraživanja, ili inovacije primjenjive na Internet i povezana računala.

sam princip rada prijenosom paketa nije mijenjan. U tom dokumentu opisane su i osnovne funkcije koje protokol mora obavljati [2][3][4][5][6] :

- Osnovni prijenos podataka (engl. *basic data transfer*): protokol ostvaruje dvosmjeran prijenos kontinuiranog niza podataka na način da niz razlomi u segmente koje pakira u pakete podataka i predaje ih protokolu IP na mrežnom sloju.
- Pouzdanost (engl. *reliability*): protokol mora omogućiti oporavak podataka koji su oštećeni, izgubljeni, udvostručeni ili isporučeni izvan redoslijeda. Pouzdanost je ostvarena:
  - dodjeljivanjem slijednog broja (engl. *sequence number*) svakom predanom oktetu i zahtjevom da primatelj potvrdi primljeni paket slanjem ACK poruke s tim slijednim brojem (engl. *Acknowledgment*),
  - ako potvrda primitka nije ostvarena u određenom vremenskom periodu, paket se ponovno šalje (engl. *Retransmission*),
  - primatelj koristi slijedni broj da segmente posloži po redoslijedu slanja i da otkrije moguće duplikate,
  - u slučaju oštećenja paketa primatelj provjerava zaštitne sume (engl. *checksum*) svakog poslanog paketa i odbacuje paket ako se sume ne podudaraju,
- Kontrola toka podataka (engl. *flow control*): protokol omogućava primatelju da javi pošiljatelju koliko podataka može primiti. To se postiže slanjem veličine "prozora" sa svakom potvrdom primitka a koji označava broj bajtova koje pošiljatelj može poslati prije slijedeće dozvole od strane primatelja.
- Multipleksiranje (engl. *Multiplexing*): protokol omogućava da istodobno istu vezu može koristiti veći broj aplikacija. To je ostvareno na način da se uvodi nova oznaka adrese, tj. broj vrata (engl. *port*). Zajedno sa IP adresom sa mrežnog sloja vrata čine adresni par koji nazivamo utičnica (engl. *socket*). Par utičnica pošiljatelja i primatelja jedinstveno označavaju svaku logičku vezu između računala. Zbog toga je moguće da istu utičnicu koristi više različitih logičkih veza.
- Upravljanje vezom: u svrhu očuvanja pouzdanosti i kontrole toka podataka, protokol zadržava informacije o svakom pojedinom toku podataka (engl. *data stream*). To

uključuje utičnice, veličinu prozora i slijednog broja što zajedno čini logičku vezu. Kad dva procesa na udaljenim računalima žele međusobno komunicirati, protokol prvo mora uspostaviti jedinstvenu vezu među njima. Po isteku komunikacije, veza se prekida.

- Sigurnost i prioriteti: posebni zahtjevi koje određuju procesi sudionici komunikacije. U odsustvu posebnih zahtjeva koriste se unaprijed zadane vrijednosti.

2.1. Struktura TCP segmenta

Svaki TCP segment se sastoji od dva dijela: zaglavlja i tijela. U zaglavlju su zapisani brojevi portova, slijedni broj i ostali upravljački sadržaji a tijelo segmenta sadrži stvarne podatke koje segment prenosi. Na (Slika 1) prikazana je struktura TCP segmenta [7].

Slika 1: Struktura TCP segmenta

Veličina zaglavlja TCP segmenta je 20 bajtova. Polja unutar segmenta su redom [2]:

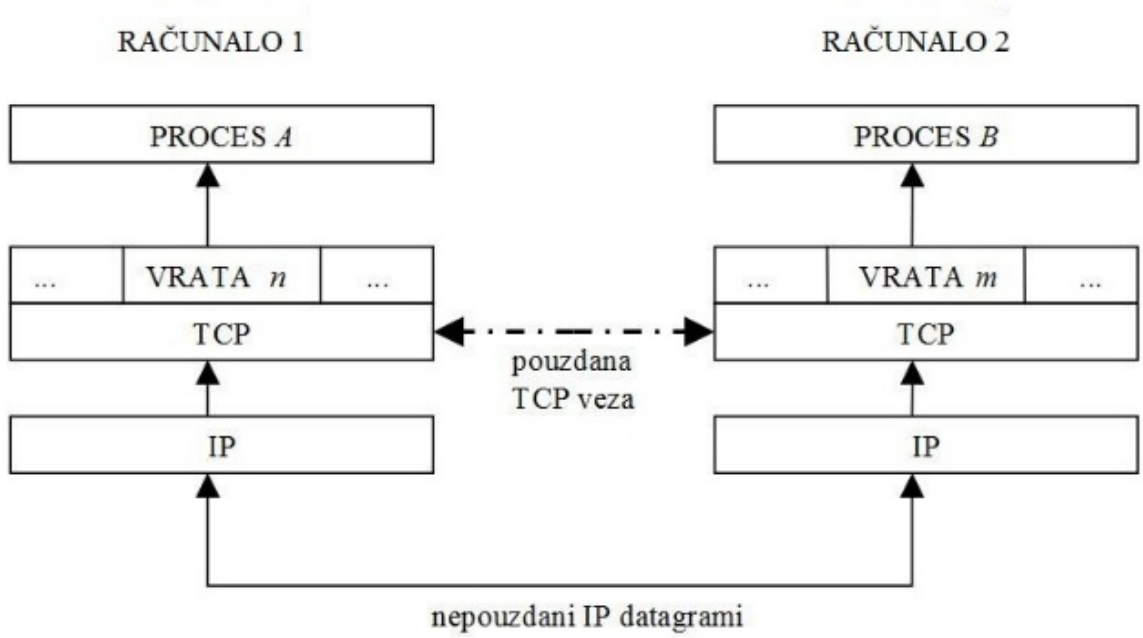
- izvorišna vrata (engl. *source port*): 16-bitni broj koji označava broj vrata procesa pošiljatelja,

- odredišna vrata (engl. *destination port*): 16-bitni broj koji označava broj vrata procesa primatelja,
- broj u nizu ili slijedni broj (engl. *sequence number*): 32-bitni broj koji označava trenutni redni broj prvog bajta podataka koji se prenose u segmentu. Izuzetno, kada opcija SYN ima vrijednost 1, u ovom polju se upisuje inicijalni slijedni broj ili ISN (engl. initial sequence number) uvećan za 1,
- broj potvrde (engl. *acknowledgment number*): 32-bitni broj koji se koristi u slučaju kada opcija ACK ima vrijednost 1, a označava slijedeći slijedni broj kojeg primatelj očekuje dobiti od pošiljatelja,
- duljina zaglavlja (engl. *header length*): u literaturi se često navodi i kao pomak podataka (engl. *data offset*) je 4-bitni broj koji označava duljinu zaglavlja. Polje opcija je neobavezno i bez njih zaglavlje je duljine 20 bajtova. Postavljanjem opcija zaglavlje može biti dulje te je nužno eksplicitno naznačiti primatelju njegovu duljinu,
- rezervirana polja (engl. *reserved*): 6-bitno polje koje se trenutno ne koristi i mora ima vrijednost 0 [2],
- upravljački bitovi (engl. *control bits*): 6-bitno polje u kojem svaki bit predstavlja zastavicu (engl. *flags*). Zastavice redom označavaju:
  - URG (engl. *Urgent Pointer*): postavljena na 1 pokazuje primatelju da mora provjeriti pokazivač hitnosti,
  - ACK (engl. *Acknowledgment*): postavljena na 1 pokazuje primatelju da mora provjeriti polje broj potvrde,
  - PSH (engl. *push*): postavljena na 1 pokazuje primatelju da se segment čim prije dostavi aplikaciji. Koristiti se za naredbe za koje nema potrebe čekati podatke prije nje, poput bezuvjetnog prekida rad (engl. *break*),
  - RST (engl. *reset*): postavljena na 1 pokazuje primatelju da pošiljatelj prekida komunikaciju. Koristi se u slučaju kad primatelj zahtjeva za otvaranje veze nije u mogućnosti uspostaviti vezu [3].
  - SYN (engl. *synchronize*): postavljena na 1 pokazuje primatelju da pošiljatelj želi uskladiti slijedne brojeve.
  - FIN (engl. *finish*): postavljena na 1 pokazuje primatelju da je pošiljatelj gotov sa prijenosom podataka i želi prekinuti uspostavljenu vezu.

- veličina prozora (engl. *window*): 16-bitni broj koji se koristi za upravljanje tokom podataka. Pomoću njega primatelj obavještava pošiljatelja koliko podataka može primiti.
- zaštitna suma (engl. *checksum*): 16-bitni broj kojeg pošiljatelj izračunava u odnosu na cijeli poslani segment a služi kao usporedni broj primatelju na osnovu kojeg može provjeriti je li došlo do greške u segmentu. Primatelj po istim pravilima izračunava zaštitnu sumu i u slučaju da se izračunata vrijednost i vrijednost ovog polja ne podudaraju, došlo je do iskrivljenja segmenta i paket se odbacuje. Ako se vrijednosti podudaraju, primatelj može potvrditi segment,
- pokazivač hitnosti (engl. *urgent pointer*): ako je zastavica URG u polju upravljačkih bitova postavljena na 1, ovo polje sadrži pokazivač na zadnji bajt hitnih podataka u segmentu koje je što je moguće prije potrebno dostaviti aplikaciji,
- opcije (engl. *options*): polje veličine do 40 bajtova koje može pružiti dodatnu funkcionalnost protokolu TCP. Polje nije obvezno i veličina mu je promjenjiva pa samim time utječe na veličinu TCP zaglavlja. Primjer korištenja opcija je obavijest pošiljatelju o najvećoj veličini segmenta kojeg primatelj može primiti, MSS (engl. *Maximum Segment Size*),
- nadopuna (engl. *padding*): ako se koriste opcije veličina zaglavlja raste, a nadopuna služi da se zaglavlje nadopuni nulama kako bi njegova veličina bila višekratnik od 32 bita,
- podaci višeg sloja (engl. *data*): polje promjenjive dužine koje sadrži podatke koji se žele prenijeti od pošiljatelja do primatelja. Polje može biti i prazno npr. kada primatelj samo želi potvrditi primitak segmenta bez slanja dodatnih podataka.

2.2. TCP logička veza

Da bi se ostvarila pouzdanost prijenosa podataka protokol TCP uspostavlja između procesa pošiljatelja i procesa primatelja logičku vezu ili kanal kako je prikazano na (Slika 2) [7].



Slika 2: Uspostavljanje logičke veze

Krajnje točke komunikacije između procesa A i B nazivaju se priključnicama (engl. *socket*). Priključnica je određena vrstom protokola transportnog sloja, IP adrese mrežnog sloja i broja vrata (engl. *port*) za svaki pojedini proces. Broj vrata je ograničen zbog veličine polja u TCP zaglavlju te se na preporuku IANA[6]-e dijele u tri grupe [8]:

- dobro poznata vrata (engl. *well-known ports*): vrata u rasponu od 0 do 1023. Koriste ih procesi koji pružaju najraširenije vrste usluga,
- registrirana vrata (engl. *registred ports*): vrata u rasponu od 1024 do 49151 registrirana pri IANA-i za određene usluge, no smiju se koristiti i u druge svrhe od strane korisnika,

6 IANA (engl. *Internet Assigned Numbers Authority*) je neprofitna organizacija zadužena za alociranje i održavanje jedinstvenih kodova i numeričkih sistema koji se primjenjuju u internetskim protokolima.

- dinamička ili privatna vrata (engl. *dynamic ports*): vrata u rasponu od 41952 do 65535 ne mogu biti registrirana i služe kao privremena vrata procesima.

Procesi koji rade na poslužiteljima za usluge uglavnom koriste ista, dobro poznata vrata. Na njima poslužitelji kreiraju prijemnu utičnicu (engl. *welcoming port*). Nakon što od strane klijenta prime zahtjev za uspostavom logičke veze, otvaraju nova nasumično odabrana vrata i kreiraju novu, utičnicu veze (engl. *connection socket*), rezerviranu za logičku vezu s klijentom koji je zahtjev uputio [5]. Time su dobro poznata vrata oslobođena za prijem novih klijenata. Popis nekih dobro znanih i rezerviranih vrata prikazan je u (Tablica 1) .

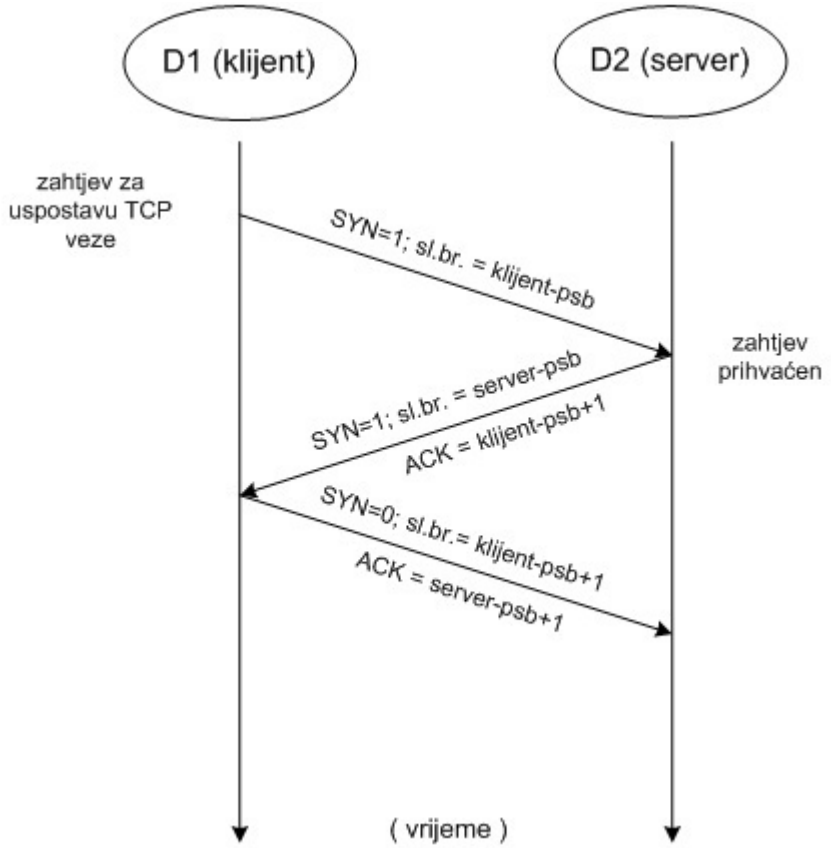
Dobro poznata vrata			
Broj vrata	Protokol	Usluga	Opis usluge
20	TCP/UDP	FTP	FTP (*File Transfer Protocol*) prijenos podataka
22	TCP/UDP	SSH	Sigurnosna asocijacija (*Secure Shell*)
25	TCP	SMTP	Razmjena elektroničke pošte (*Simple Mail Transfer Protocol*)
80	TCP	HTTP	Usluga prikaza mrežnih stranica (*Hypertext Transfer Protocol*)
Rezervirana vrata			
1194	TCP/UDP	OpenVPN	*Open source* virtualna privatna mreža
3306	TCP/UDP	MySQL	Vrata rezervirana za MySQL bazu
1214	TCP	Kazaa	Kazaa mrežni servis za razmjenu sadržaja
16567	UDP	Battlefield 2	Vrata rezervirana za igru Battlefield 2
31457	TCP	TetriNET	Vrata rezervirana za mrežni Tetris

Tablica 1: Primjeri dobro poznatih i rezerviranih vrata

Logička veze ili kanal je virtualna veza između dvije krajnje točke komunikacije ali to ne znači da će svi paketi biti usmjeravani istim putem kroz mrežu. Svaka veza prolazi kroz tri faze: uspostavljanje veze, prijenos podataka i prekid veze.

2.2.1. Uspostavljanje TCP veze

Dva procesa uspostavljaju vezu razmjenom poruka kako prikazuje (Slika 3) [5]. Ovaj način uspostave veze se naziva i rukovanjem u tri koraka ili trostrukim rukovanjem (engl. *three-way handshake*).



Slika 3: Uspostavljanje TCP veze (trostruko rukovanje)

U prvom koraku klijent (D1) koji želi komunicirati sa serverom (D2) šalje jedan segment u kojem je u upravljačkom polju zaglavlja zastavica SYN postavljena na vrijednost 1, te se taj segment naziva SYN segment [3]. U zaglavlje SYN segmenta, klijent metodom slučajnog odabira upisuje svoj početni slijedni broj ili ISN (engl. *Initial Sequence Number*) u polje slijednog broja i predaje segment mrežnom sloju (IP protokolu) koji segment pakira i prenosi poslužitelju D2.

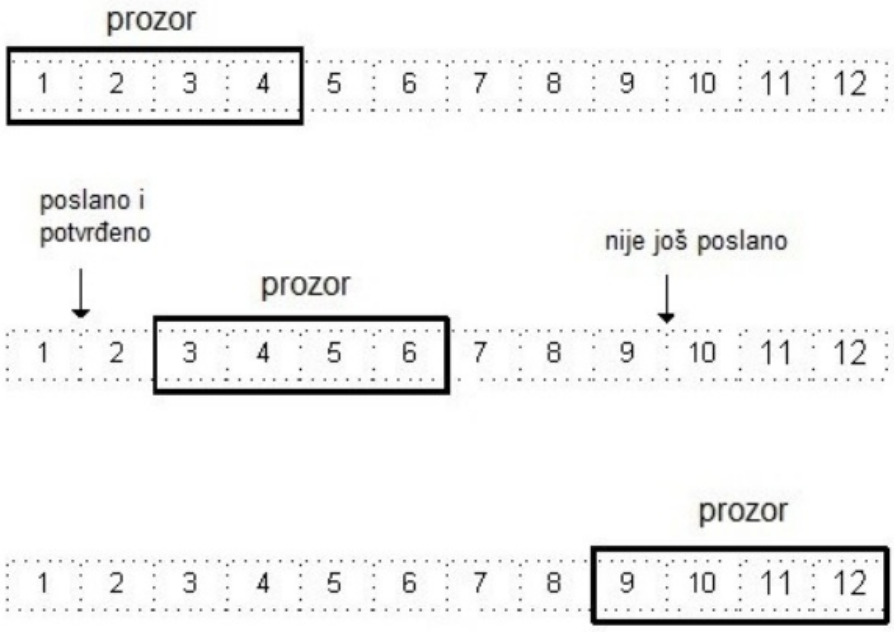
U drugom koraku poslužitelj D2 prima SYN segment klijenta D1 koji traži uspostavu veze i dodijeljuje memorijske prostore za prijem i slanje podataka. Kao odgovor priprema

upravljački segment koji ne sadrži korisničke podatke. U zaglavlju segmenta, u polju zastavica, postavlja SYN na vrijednost 1. U polje potvrde primitka upisuje vrijednost slijednog broja kojeg je dobio od klijenta uvećanu za 1, te u polje početni slijedni broj upisuje slučajno odabrani svoj slijedni broj. Ovakav segment se naziva SYNACK segmentom [3], odnosno potvrdom SYN segmenta i njime poslužitelj D2 javlja klijentu D1 da je prihvatio zahtjev za uspostavu veze sa njegovim slijednim brojem.

U trećem koraku klijent prima SYNACK segment sa poslužitelja te također dodijeljuje memorijski prostor za primanje i slanje podataka. Početni slijedni broj koji je dobio od poslužitelja uvećava za 1 i upisuje u polje potvrde primitka, čime potvrđuje primitak SYNACK segmenta. Priprema novi segment u kojemu je SYN zastavica postavljena na 0 jer se smatra da je trećim korakom logička veza uspostavljena. U ovom segmentu klijent može početi slati podatke poslužitelju. Nakon uspostave veze oba računala mogu slati podatke u oba smjera istodobno.

2.2.2. Prijenos podataka

Pri prijenosu podataka može doći do pogreški. Neki paketi mogu biti izgubljeni po putu, doći do primatelja izmjenjeni ili izvan poslanog redoslijeda. Primatelj redovno izvještava pošiljatelja o uspješno primljenim segmentima slanjem slijednog broja segmenta u polju potvrde primitka. Ukoliko nakon određenog vremena potvrda ne stigne do pošiljatelja, segment se smatra izgubljenim i pošiljatelj ga ponovno šalje (retransmisija). Postoje dvije metode slanja segmenata: stani i čekaj (engl. *stop and wait*) i punjenje cjevovoda (engl. *pipelining*) [3]. Metodom stani i čekaj pošiljatelj šalje jedan segment i čeka njegovu potvrdu prije slanja slijedećeg segmenta. Kod punjenja cjevovoda, pošiljatelj šalje veći broj segmenata bez da čeka potvrdu njihova primitka. Segmenti imaju svoje slijedne brojeve koji su neprekinut rastući niz cijelih brojeva u intervalu $[0, 2^{n-1}]$, gdje je n dužina polja slijednog broja u bitovima, što u protokolu TCP iznosi 16. Broj nepotvrđenih segmenata koje pošiljatelj smije imati na putu predstavlja okvir ili prozor na taj niz koji se stalno pomiče (klizi) prema gore i naziva se kliznim ili klizećim prozorom (engl. *sliding window*)[3]. Primatelj čeka na potvrdu poslanih segmenata iz prozora i zadnji potvrđeni segment predstavlja lijevu granicu kliznog prozora kako je prikazano na (Slika 4) [7].



Slika 4: Klizni prozor (sliding window)

Protokol TCP koristi metodu kumulativnog potvrđivanja, što znači da potvrda primitka nekog segmenta ujedno označava i potvrdu primitka svih segmenata prije njega.

Vrijeme čekanja na potvrdu izračunava se na osnovu povremenih stvarnih mjerenja vremena povratnog puta - RTT (engl. *round trip time*). RTT varira ovisno o opterećenju mreže i usmjerivača preko kojih logička veze prolazi i stoga je potrebno izvršiti optimiziranje vremena čekanja prije slanja svakog segmenta. Na početku TCP procjenjuje RTT za vezu, a zatim mjeri vrijeme od trenutka slanja segmenta do trenutka kad primi potvrdu o njegovom primitku. Na temelju tih vrijednosti izračunava se nova vrijednost procjenjenog RTT-a prema izrazu (1) [5]:

$$RTT(t) \leq (1-\alpha) * RTT(t-1) + \alpha * IzRTT \qquad (1)$$

gdje je:

- RTT(*t*): nova procijenjena vrijednost vremena povratnog puta,
- RTT(*t*-1) : prethodna procijenjena vrijednost vremena povratnog puta,

– IzRTT: izmjerena vrijednost vremena povratnog puta,

– α: koeficijent iz intervala [0<α<1].

Ako je koeficijent α bliže vrijednosti 1, na novu procijenjenu vrijednost veći utjecaj ima izmjerena vrijednost. Najčešće se za α uzima vrijednost 1/8 čime se postiže da procijenjene vrijednosti nemaju nagle skokove, jer ih čini 7/8 prethodne vrijednosti i samo 1/8 trenutno izmjerene vrijednosti. Pored mjerenja i izračunavanja procijenjenog vremena RTT-a, izračunava se i veličina odstupanja mjerenog vremena od procijenjenih vrijednosti prema izrazu (2) [5]:

$$DevRTT \leq (1-\beta) * DevRTT + \beta * |RTT(t) - IzRTT| \qquad (2)$$

gdje je:

– DevRTT: veličina odstupanja (devijacije) RTT-a,

– β : koeficijent iz intervala [0,1] za koji se najčešće uzima vrijednost 1/4.

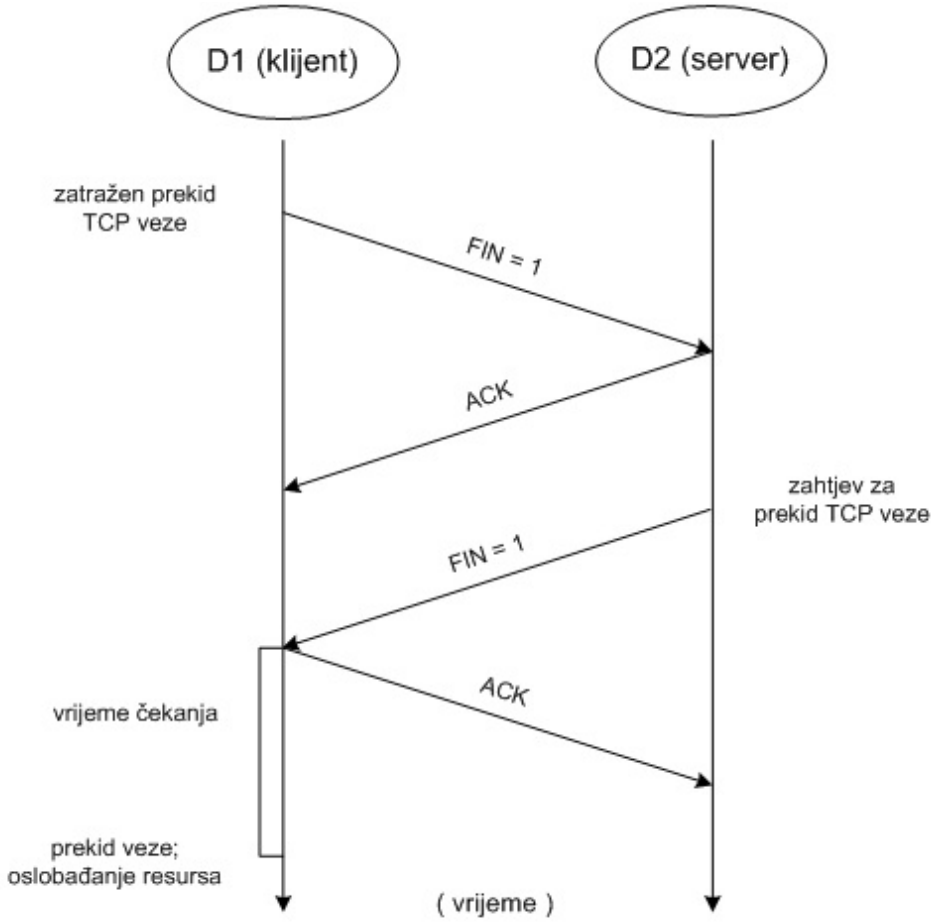
Ako je vrijednost RTT relativno stabilna tokom trajanja veze, razlike između procijenjenih i izmjerenih vremena, pa tako i veličina odstupanja, će biti male. Protokol na temelju procijenjenog vremena povratnog puta i njegovog odstupanja od izmjerene vrijednosti izračunava vrijeme čekanja (RTO, engl. *retransmission timeout*) na potvrdu segmenta prema izrazu (3) [5]:

$$RTO = RTT(t) + 4\,x\,DevRTT \qquad (3)$$

Ovi izračuni se zasnivaju na iskustvu, promatranjima i mjerenjima te se mogu izmjeniti odabirom drugih vrijednosti koeficijenata α i β, ili odabirom druge vrijednosti umjesto 4 [3]. Kad vrijeme čekanja na potvrdu istekne, protokol TCP ponavlja slanje najstarijeg nepotvrđenog segmenta i iznova pokreće odbrojavanje vremena.

2.2.3. Raskidanje TCP veze

Raskid veze može zatražiti bilo koji od procesa koje ta veza povezuje. Raskid veze je važan da bi se oslobodili resursi dodijeljeni vezi (memorijski prostori) na oba računala. Na (Slika 5) prikazan je postupak raskidanja veze koji pokreće klijent [5].



Slika 5: Raskidanje TCP veze

Klijent D1 u polju zastavica postavlja zastavicu FIN na vrijednost 1 čime obavještava primatelja da je aplikacijski proces zatražio prekid TCP veze. Kad poslužitelj D2 zaprimi zahtjev, klijentu šalje potvrdu primitka. U slijedećem segmentu poslužitelj u zaglavlju postavlja zastavicu FIN na vrijednost 1 i šalje ga klijentu. Klijent po primanju potvrđuje primljeni segment i time je veza raskinuta, te oba računala oslobađaju alocirane resurse veze.

2.3. Upravljanje tokom podataka

Pri uspostavljanju TCP veze, računala sudionici veze dodjeljuju dio memorije koja služi za prijem i slanje segmenta. Kada računalo primi ispravan segment, onda se segment zapisuje u dodijeljeni dio memorije odakle ga preuzima proces kojem je segment namijenjen. Taj dio dodijeljene memorije se naziva prijemnom memorijom ili prijemnim baferom (engl. *Buffer*) [4] [5]. Iz prijemne memorije proces preuzima segment na obradu i potom ga briše. Kako je vremenski period obrade segmenta različit i ponekad duže traje s obzirom na zauzetost procesa drugim poslovima, može se dogoditi da se prijemna memorija ispuni segmentima koji nisu obrađeni. Zbog nemogućnosti da zapiše daljnje segmente, primatelj odbacuje nove segmente sve dok se memorijski prostor ne oslobodi. Kako se to ne bi događalo, protokol TCP sadrži metodu upravljanja tokom podataka koji usklađuje brzinu slanja podataka pošiljatelja s brzinom prijema i obrade od strane primatelja na način da primatelj obavještava pošiljatelja o količini podataka koje može primiti.

Prijemna memorija je podijeljena u dva dijela; dio koji je ispunjen pristiglim ali još nepotvrđenim segmentima, i dio koji je prazan. Razlika između ukupno dodijeljenog memorijskog prostora i prostora zauzetog nepotvrđenim segmentima predstavlja veličinu prozora primatelja. Podatak o veličini prozora primatelj zapisuje u zaglavlje TCP segmenta u svakoj potvrdi primljenih segmenata. Na taj način pošiljatelj može prilagođavati brzinu slanja segmenta da bi se izbjeglo odbacivanje paketa.

Poseban slučaj se događa u trenutku kada broj primljenih ali ne potvrđenih segmenta ispuni prijemnu memoriju primatelja, odnosno da je veličina njegovog prijemnog prozora jednaka 0. Pošiljatelj, koji je dobio poruku da primatelj nema više mjesta za prijem, prestaje sa slanjem daljnjih segmenta. U međuvremenu proces na primatelju obradi segmente iz prijemne memorije i oslobodi prostor, ali, kako su ti segmenti već ranije potvrđeni, ne obavještava pošiljatelja o promjeni.

Protokol ovaj problem rješava na način da pošiljatelj nastavi sa slanjem segmenata veličine jednog bajta i nakon što je od primatelja primio obavijest o zauzeću prijemne memorije. Ti segmenti će biti odbačeni dok god je prijemni prozor primatelja pun, ali čim se prozor otvori primatelj će u potvrdi tog jednobajtnog segmenta obavijestiti pošiljatelja o novoj veličini prozora [5].

2.4. Upravljanje zagušenjem mreže

Upravljanje tokom podataka uspješno usklađuje komunikaciiju dvaju procesa na krajnjim točkama veze, no nemaju informaciju o opterećenosti usmjerivača između njih, u unutrašnjosti veze. Porastom prometa preko usmjerivača može se dogoditi da usmjerivač ne može proslijediti dolazne pakete[7] te ih odbacuje. Odbacivanjem paketa na usmjerivaču, poslani segmenti su izgubljeni i ne dolaze do primatelja, a time izostaje i potvrda primitka.

Vrijeme čekanja na potvrdu (RTO) pošiljatelja istječe i on ponovno šalje segment, dodatno opterećujući usmjerivač koji paket ponovno odbacuje. To dovodi do značajnog pada učinkovitosti mreže, odnosno do zagušenja mreže. Otkrivanje i otklanjanje zagušenja može se izvesti samo na transportnoj razini (s ruba mreže), ili na transportnoj razini ali uz izravnu poruku usmjerivača. Na Internetu, problem zagušenja se riješava samo na transportnoj razini protokolom TCP[8] [5].

Uz veličinu prijemnog prozora primatelja kojeg neposredno saznaje iz zaglavlja potvrdne poruke, TCP pošiljatelja pokušava utvrditi mogućnosti prijenosa usmjerivača preko kojih se odvija logička veza i to na temelju uspješnosti prijenosa vezom. Protokol na primatelju uvodi veličinu koja se naziva prozor zagušenja (engl. *congestion window*) a koja pokazuje koliko segmenata pošiljatelj može imati s obzirom na njegovu procjenu prijenosa usmjerivača. Procjena se temelji na kretanju veličine RTT i broja izgubljenih segmenta.

Gubitak segmenta se obično događa zbog odbacivanja IP paketa na nekom usmjerivaču unutar mreže koji je dostigao stanje zagušenja. Pošiljatelj tada treba smanjiti veličinu svog prozora zagušenja. S druge strane, svaki dolazak nove potvrde o uspješno primljenom segmentu sa primateljeve strane ukazuje da mreža nije zagušena i da pošiljatelj može pokušati povećati broj segmenta, odnosno svoj prozor zagušenja, da bi se optimalno iskoristio kapacitet prijenosa mreže.

Stanje mreže i opterećenost (zagušenost) usmjerivača unutar mreže se stalno mijenja i protokol TCP pokušava otkriti intezitet slanja tako da za svaki uspješno primljeni ACK linearno povećava prozor zagušenja, a za svaki gubitak segmenta oštro smanjuje prozor zagušenja.

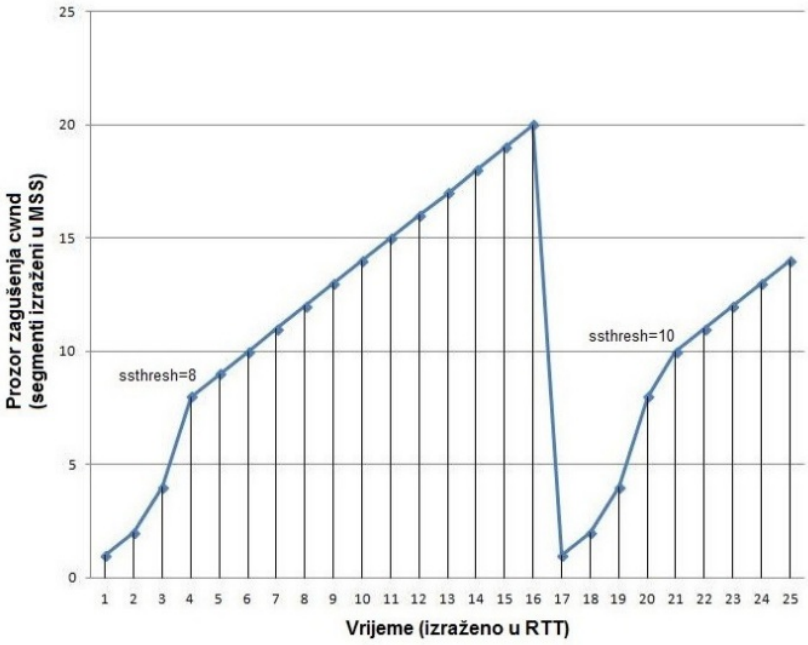
7 IP paketi mrežnog sloja
8 protokol UDP, iako na transportnoj razini, ne vodi računa o zagušenju mreže

Protokol TCP koristi četiri metode kojima pokušava otkloniti zagušenje na usmjerivačima unutar veze. To su: spori početak (engl. *slow start*), izbjegavanje zagušenja (engl. *congestion avoidance*), brzo ponovno slanje (engl. *fast retransmit*) i brzi oporavak (engl. *fast recovery*) [4] [9].

2.4.1. Spori početak i izbjegavanje zagušenja

Pri uspostavi TCP logičke veze veličina prozora zagušenja postavlja se na jedan MSS[9], što znači da pošiljatelj tom vezom šalje samo jedan segment i čeka na povratnu potvrdu od primatelja. Pošiljatelj želi što prije dostići najveću brzinu koju mreža u datom trenutku dopušta te za svaku primljenu potvrdu primitka (ACK) šalje dva nova segmenta. Time se broj poslanih segmenata u svakom ciklusu RTT-a udvostručuje, sve dok se ne dogodi gubitak segmenta.

Ako je gubitak segmenta otkriven na način da je isteklo vrijeme čekanja na potvrdu segmenta, TCP pošiljatelja zapamti vrijednost polovice dosegnute vrijednosti prozora zagušenja. Ta vrijednost predstavlja prag sporog početka (engl. *slow start threshold, ssthresh*). Novu vrijednost prozora zagušenja vraća na jedan segment i ponovno šalje izgubljeni segment. Protokol pritom povećava veličinu prozora zagušenja kao i na početku veze sve dok njegova veličina ne dosegne prag sporog početka, *ssthresh*.



Slika 6: Spori početak i izbjegavanje zagušenja

9 MSS (engl. *Maximal Segment Size*) je najveća dopuštena dužina segmenta.

Po završetku sporog početka, veličina prozora zagušenja je postavljena na polovicu veličine prozora prije gubitka paketa. Protokol prekida udvostručavanje veličine prozora jer bi to dovelo do ponovnog zagušenja i odbacivanja paketa. Umjesto toga prozor se povećava linearno, za jedan MSS svaki RTT, sve dok ne dođe do ponovnog odbacivanja paketa na zagušenom usmjerivaču (Slika 6) [7].

Ako i na ovaj gubitak segmenata ukazuje istek vremena čekanja, TCP postupa kao i kod sporog početka: postavlja veličinu prozora zagušenja na jedan MSS, postavlja prag sporog početka (*ssthresh*) na polovinu postignute vrijednosti prozora zagušenja i vraća se u proces sporog početka slanjem po dva segmenta za svaku primljenu potvrdu primitka.

Ako je gubitak segmenta utvrđen primanjem više od tri potvrdne poruke koje potvrđuju segment s istim slijednim brojem, to ukazuje da zagušenje postoji ali nije potpuno, jer su dva segmenta uspješno stigla do primatelja nakon izgubljenog segmenta. U ovom slučaju veličina prozora zagušenja se ne mora smanjiti na jedan MSS već može iznositi polovinu vrijednosti prozora u trenutku kad je došlo do gubitka. Nakon toga protokol povećava prozor za jedan MSS svaki ciklus RTT-a.

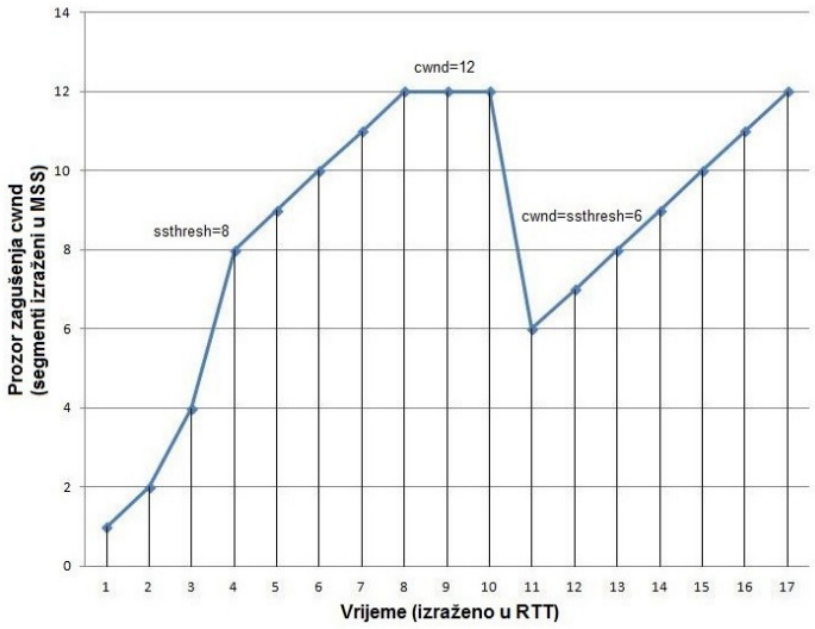
Brzi početak i izbjegavanje zagušenja su obavezne metode koje moraju biti uključene u sve verzije TCP protokola [10].

2.4.2. Brzo ponovno slanje i brzi oporavak

Kako se vrijeme čekanja na istek vremena potrebnog za prijem potvrde segmenta (RTO) udvostručava nakon izgubljenog segmenta, intezitet prijenosa vezom s čestim gubitcima segmenata naglo opada iz razloga što pošiljatelj može dugo čekati na istek RTO-a. Zbog toga se protokol može modificirati na način da u slučaju kad postoji indikacija gubitka segmenta, pošiljatelj ne čeka istek vremena već odmah šalje izgubljeni segment. Ova metoda se naziva brzo ponovno slanje ili brza retransmisija (engl. *fast retransmit*) (Slika 7). Dovoljan pokazatelj gubitka segmenta je kad pošiljatelj primi više potvrda primitka za isti segment od strane primatelja.

Više potvrda za isti segment nastaju kad primatelj dobije segmente nakon izgubljenog paketa, jer za svaki primljeni segment šalje potvrdu o primitku ali sa slijednim brojem izgubljenog segmenta. To su dvostruke potvrde ili DUPACK (engl. *duplicate acknowledgment*), a njihov primitak znači ili gubitak segmenta ili da je segment isporučen izvan redoslijeda. TCP pošiljatelja pokreće brzu retransmisiju po primitku tri dvostruke

poruke za redom. Po pokretanju brze retransmisije, pošiljatelj prelazi u fazu sporog početka kao u slučaju kada dođe do isteka RTO-a. Iskustvo je pokazalo da zbog toga dolazi do naglog pada brzine prijenosa samo zbog jednog izgubljenog segmenta [5]. Zbog toga se u kombinaciji sa brzim ponovnim slanjem koristi i brzi oporavak (engl. *fast recovery*), koji omogućava brži oporavak brzine prijenosa kada pošiljatelj primi više od tri iste potvrde od primatelja.



Slika 7: Brzo ponovno slanje i brzi oporavak

Po primitku tri potvrde istog segmenta metoda brzog oporavka u protokolu TCP prvo izračuna prag sporog početka *sstresh* tako da ga postavlja na polovinu prozora zagušenja prije samog zagušenja ili na vrijednost dva MSS [RFC 2001], ovisno o tome koja je od tih dvaju veličina veća, prema izrazu (4):

$$sstresh = max\left\{\frac{cwnd}{2}, 2xMSS\right\} \qquad (4)$$

Nakon toga obavi se brzo ponovno slanje izgubljenog segmenta i poveća prozor zagušenja *cwnd* za 1 MSS za svaku od tri primljene dvostruke potvrde, prema izrazu (5):

$$cwnd = sstresh + 3 * MSS \qquad (5)$$

Primitak nove potvrde dolaska novih segmenata do primatelja označava da je zagušenje usmjerivača unutar mreže prestalo. Vrijednost prozora zagušenja *cwnd* se postavlja na vrijednost praga sporog početka i prelazi se u fazu izbjegavanja zagušenja.

2.4.3. Implementacije protokola TCP

Postoji više različitih implementacija protokola TCP [3]. Prema RFC 1122 [10] sve implementacije moraju podržavati metode sporog početaka i izbjegavanja zagušenja. Implementacija *TCP Tahoe*, prvotno objavljena u 4.3BSD[10] Tahoe, uz njih koristi i brzo ponovno slanje kojeg prati spori početak. Algoritam brzog oporavka dodan je *TCP Reno* implementaciji, prvotno objavljenoj u 4.3BSD Reno distribuciji.

Implementacija *New Reno* [11] poboljšava brzi oporavak u odnosu na *TCP Reno*. *Reno* svakim gubitkom segmenta smanjuje prozor zagušenja *cwnd* na polovinu vrijednosti po ulasku u brzi oporavak, što u slučaju gubitaka više segmenata unutar istog prozora rezultira značajnim smanjenjem prozora. Po primitku dvostruke potvrde *New Reno* prelazi u brzi oporavak, ali, za razliku od implementacije *Reno,* ne izlazi iz brzog oporavka sve dok segmenti poslani do tog trenutka ne budu potvrđeni.

Po ulasku u fazu brzog oporavka *New Reno* zapamti broj poslanih segmenata i po primitku slijedeće potvrde koja potvrđuje sve segmente poslane do trenutka ulaska u brzi oporavak, postavlja prozor zagušenja *cwnd* na vrijednost praga sporog početka *sstresh* i nastavlja sa izbjegavanjem zagušenja. Međutim, ukoliko primljena potvrda ne potvrđuje sve segmente koji su poslani do početka brzog oporavka, zaključuje da je i slijedeći segment izgubljen, ponovno ga šalje i postavlja broj dvostrukih potvrda na nulu. Iz brzog oporavka izlazi nakon što su svi poslani segmenti potvrđeni.

Implementacija *TCP Vegas* uvodi novi način predviđanja zagušenja prije samog gubitka segmenta. *Vegas* pamti vrijednost sata za svaki poslani segment i po primitku potvrde od primatelja računa novu vrijednost RTT-a, ali sa većom preciznošću (zrnatosti sata) od implementacije *Reno*. Pomoću takve preciznije vrijednosti izračuna RTO za svaki poslani segment i u slučaju primitka dvostruke potvrde provjerava ga i, ako je RTO istekao, ponovno šalje segment bez čekanja na primitak još dvije dvostruke potvrde.

Glavna prednost implementacije *Vegas* u odnosu na *Reno* je predviđanje zagušenja na temelju propusnosti mreže još u fazi izbjegavanja zagušenja. Očekivana propusnost se računa po izrazu (6), pri čemu je RTT_{base} vrijeme povratnog puta za nezagušenu mrežu[11]:

10 engl. *Berkeley Software Distribution* inačica je UNIX-a razvijena sedamdesetih godina 20. st. na University of California, Berkeley. Inačice izvedene iz nje se nazivaju BSD-distribucije.

11 U praksi to je najmanja izmjerena vrijednost RTT-a

$$P(očekivano) = \frac{cwnd}{RTT_{base}} \qquad (6)$$

Stvarna propusnost se računa svaki RTT kao omjer poslanih segmenata (*N*) unutar jednog RTT-a i vrijednosti tog RTT-a po izrazu (7):

$$P_{stvarno} = \frac{N}{RTT} \qquad (7)$$

Izračunata razlika ovih dvaju veličina uspoređuje se s zadanim pragovima koji se odnose na premali/preveliki broj dodatnih podataka unutar mreže. U slučaju kad je razlika mala *Vegas* će linearno povećati *cwnd* u slijedećem RTT-u. Suprotno tomu, ako razlika postane veća od gornjeg praga, *cwnd* će se linearno smanjivati. Razlika unutar granica neće utjecati na veličinu prozora zagušenja. Promjene u odnosu na *Reno* postoje i pri sporom početku gdje *Vegas* također procijenjuje raspoloživu propusnost unutar mreže i eksponencijalno povećava veličinu prozora zagušenja svaki drugi RTT. Kad vrijednost $P_{stvarno}$ postane manja od vrijednosti $P_{očekivano}$ prelazi u izbjegavanje zagušenja.

TCP SACK (engl. *Selective Acknowledgment*) [SACK] je implementacija protokola koja omogućava otkrivanje gubitka i ponovno slanje više segmenata unutar jednog RTT-a. Uvodi novu metodu potvrđivanja primljenih segmenata razmjenom dodatne informacije koju zapisuje u neobavezno polje opcija u zaglavlju segmenta. Korištenje ove metode je moguće samo ako oba kraja TCP logičke veze podržavaju implementaciju, što se mora eksplicitno odrediti pri uspostavi veze. Različite implementacije protokola su brojne, a neke od njih su navedene u (Tablica 2).

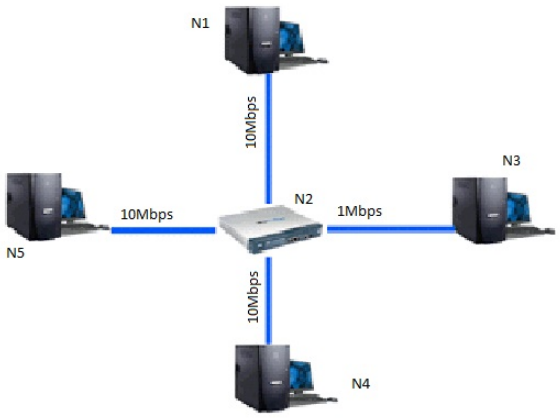
TCP Hybla	FAST TCP	Westwood	TCP Illinois
TCP BIC	H-TCP	Westwood+	TCP Ghergani
TCP CUBIC	Data Center TCP	Compound TCP	TCP PRP

Tablica 2: Implementacije TCP protokola

3. Simulacija mreže

Topologija simulirane mreže, prikazana na (Slika 8), je oblika zvijezde sa četiri računala i jednim usmjerivačem. Sve simulacije su izvedene korištenjem simulatora Ns-3, dok je graf funkcije promjene veličine prozora zagušenja *cwnd* izveden uz pomoć *gnuplot* aplikacije.

Cilj simulacije je usporediti brzine prijenosa podataka i veličine prozora zagušenja, te analizirati ponašanje implementacija TCP protokola (*TCP Tahoe, TCP Reno* i *TCP NewReno*) pri zagušenju mreže kroz tri različite situacije mrežnog prometa, gledano sa strane procesa korisnika na računalu N1 koji uspostavlja vezu sa procesom na računalu N3 preko usmjerivača N2. Usko grlo mreže, na kojem se očekuje zagušenje pri porastu prometa mrežom, predstavlja veza od usmjerivača N2 do računala N3, dok sam usmjerivač N2 ima postavljen red čekanja na 25 paketa nakon čega počinje odbacivati pristigle pakete. Prva simulirana situacija prikazuje kako se ponašaju različite implementacije TCP-a kad na usmjerivaču dođe do odbacivanja paketa pri neprekidnom prijenosu podataka vezom N1-N3 u trajanju 30 s. Druga situacija prikazuje ponašanje kada 10 s nakon što je N1 počeo slati podatke N3, računalo N4 uspostavlja vezu sa N3 i šalje podatke u trajanju od 10 s koristeći TCP protokol, dok u trećoj simuliranoj situaciji računalo N5 uspostavlja vezu sa N3, ali pri tome za prijenos podataka koristi protokol UDP.



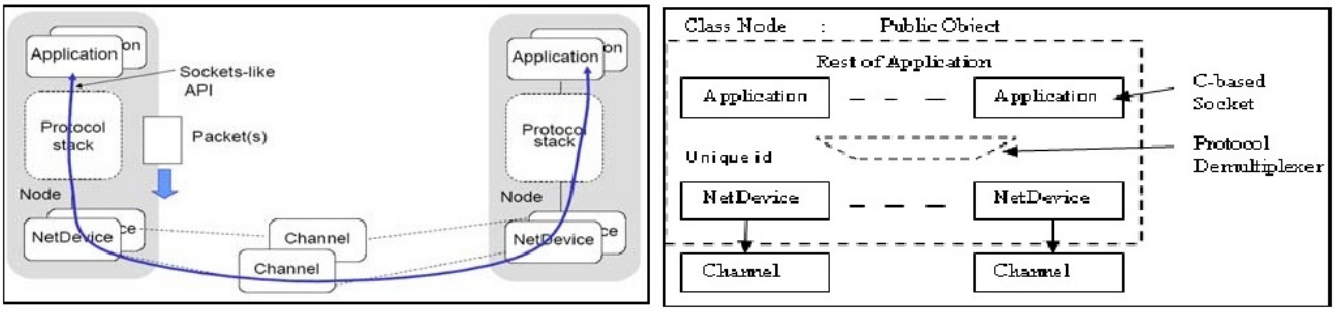
Slika 8: Topologija simulirane mreže

3.1. Simulator računalne mreže Ns-3

Simulator računalne mreže NS-3 (engl. *Network simulator version 3*) je *open-source* aplikacija razvijena u programskom jeziku *C++*. To je skup programskih biblioteka[12] (engl. *library)* koje se mogu statički ili dinamički povezati sa *main* funkcijom programa napisanog u *C++*-u unutar kojeg se definira topologija simulirane mreže i pokreće sama simulacija. Slično tomu, cjelokupno aplikacijsko programsko sučelje (API)[13] može se uvesti i u *Phyton* kao "ns3" modul. Simulacije nisu u stvarnom vremenu već su pogonjene određenim diskretnim događajima (engl. *event driven*). Objekti (klase) kojima se ostvaruje simulacija su [12] [13] [14]:

- Čvor (engl. *Node*): predstavlja matičnu ploču sa memorijom, procesorom i ulazno-izlaznim jedinicama. Može biti računalo domaćin (engl. *host*) ili usmjerivač (engl. *router*) unutar mreže.
- Aplikacija (engl. *Application*): generator paketa na čvoru unutar mreže. Zadaća je simulirati procese koji ostvaruju vezu na mreži.
- Mrežni uređaji (engl. *Net device*): su mrežne kartice postavljene na čvorovima i omogućuju komunikaciju sa drugim čvorovima simulirane mreže.
- Kanali (engl. *Channel*): simulirane fizičke spone između različitih mrežnih uređaja.
- Paketi (engl. *Packet*)
- Utičnice (engl. *Socket*)
- Spremnici (engl. *Containers*) i pomoćnici (engl. *Helpers*)

Osnovna arhitektura simulatora i arhitektura pojedinog čvora prikazane su na (Slika 9)[12].



Slika 9: Ns-3 osnovna i arhitektura čvora

Ns-3 prati četveroslojni model računalne mreže. Modeli koje podržava prikazani su u (Tablica 3)[12].

	Ns-3 modeli
Sloj aplikacija	Ping, vat, telnet, FTP, multicast FTP, HTTP, webcache, peer-to-peer
Prijenosni sloj	TCP (više implementacija), UDP, SCTP, XCP, TFRC, RAP, RTP, PGM, RLM, PLM, DCCP
Mrežni sloj	IPv4, IPv6, Mobile IP, IpinIP, Nixvector, SRM, DSR, DSDV, TORA, IMEP, NAT, BGP, OSPF, RIP, IS-IS, PIM-SM
Fizički sloj	Two-way, Satellite repeater, Omni Antennas (wifi), GSM

Tablica 3: Simulacijski modeli Ns-3

Ns-3 je zamišljen kao nasljednik simulatora mreže Ns-2, no odlučeno je da će biti iznova dizajniran i napisan, stoga nije kompatibilan sa prethodnikom. Tijekom razvoja pojedini dijelovi dizajna i koda preuzeti su iz simulatora *GTNetS*, *yans* i Ns-2. Cilj razvoja je stvoriti simulator mreže koji će se nastaviti razvijati od strane akademske zajednice i zainteresiranih tvrtki. Oko projekta je okupljena zajednica održavatelja (engl. *maintainers*) od kojih svatko održava i osvježava dio koda simulatora. Kod mrežnog simulatora ns-3 u potpunosti je dostupan licencom GPLv2[14].

14 *GNU General Public License* (kratice GNU GPL i samo GPL) je vjerojatno najpoznatija i najšire korištena licenca za slobodan softver, koju je originalno kreirao Richard Stallman za projekt GNU, a o kojoj se danas brine *Free software foundation* (FSF)

3.2. Prikupljanje i prikaz rezultata simulacije

Za izradu simulacije i izradu C++ *main* programa korišteno je programsko okruženje *geany.* Povezivanje sa modulima simulatora Ns-3 obavljeno je naredbom `#include<>`. Izbor implementacije TCP-a je obavljeno prije stvaranja objekata simulacije, proslijeđivanjem parametra `ns3::TcpL4Protocol::SocketType` statičkoj klasi `Config`.

```
Config::SetDefault ("ns3::TcpL4Protocol::SocketType", StringValue
("ns3::TcpTahoe"));
```

Podsustav logiranja simulacijskih događaja korišten je da bi se prikupili podatci o trajanju prijenosa i količini prenesenih podataka.

```
LogComponentEnable ("PacketSink", LOG_LEVEL_INFO);
```

Odabrana topologija sastoji se od pet čvorova povezanih u oblik zvijezde. Metodom `Create()` stvoreni su u instanci `allnodes` klase `NodeContainer`, te metodom `Get()` postavljeni na instance koje prate oblik mreže.

```
NodeContainer allNodes, nodes12, nodes23, nodes24,nodes25;
  allNodes.Create (5);
  nodes12.Add (allNodes.Get (0));
  nodes12.Add (allNodes.Get (1));
  nodes23.Add (allNodes.Get (1));
  nodes23.Add (allNodes.Get (2));
  nodes24.Add (allNodes.Get (1));
  nodes24.Add (allNodes.Get (3));
  nodes25.Add (allNodes.Get (1));
  nodes25.Add (allNodes.Get (4));
```

Pomoćnik `PointToPointHelper` [15] stvara mrežne uređaje, dodaje ih čvorovima i povezuje logičkom mrežom. Metodom `SetDeviceAttribute()` postavljene su vrijednosti brzine prijenosa podataka (10Mbps) i vrijeme kašnjenja (10ms) za veze između čvorova mreže, osim između usmjerivača i čvora 3 gdje je brzina veze smanjena na 1Mbps.

```
PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("10ms"));

NetDeviceContainer devices12, devices23, devices24,devices25;
  devices12 = pointToPoint.Install (nodes12);
  devices24 = pointToPoint.Install (nodes24);
  devices25 = pointToPoint.Install (nodes25);

  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("1Mbps"));
  devices23 = pointToPoint.Install (nodes23);
```

Ovim je postignuto usko grlo mreže i na usmjerivaču će dolaziti do odbacivanja paketa. Pri tom, na usmjerivaču postoji određeni broj paketa koji čekaju u redu na usmjeravanje prije nego što budu odbačeni. Najjednostavniji red čekanja je objekt `DropTailQueue` [16] na kojem je postavljen oblik odbacivanja (`QUEUE_MODE_PACKETS`) i najveći broj paketa u redu (25).

```
  Ptr<DropTailQueue> queue = CreateObject<DropTailQueue> ();
  queue->SetAttribute ("Mode", EnumValue (DropTailQueue::QUEUE_MODE_PACKETS));
  queue->SetAttribute ("MaxPackets", UintegerValue (25));
  queue->TraceConnectWithoutContext ("Drop", MakeCallback (&QueueTailDrop));

  devices23.Get (0)->SetAttribute ("TxQueue", PointerValue (queue));
```

Slijedi dio dodijeljivanja TCP/IP funkcionalnosti svim čvorovima i određivanja IP adresa čvorova [17] [18]:

```
  InternetStackHelper stack;
  stack.Install (allNodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer interfaces12 = address.Assign (devices12);

  address.SetBase ("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer interfaces23 = address.Assign (devices23);

  address.SetBase ("10.1.3.0", "255.255.255.0");
  Ipv4InterfaceContainer interfaces24 = address.Assign (devices24);

  address.SetBase ("10.1.4.0", "255.255.255.0");
  Ipv4InterfaceContainer interfaces25 = address.Assign (devices25);


  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Funkciju pošiljatelja obavljaju *on-off* aplikacije instalirane na različitim čvorovima ovisno o simuliranim situacijama. Funkciju primatelja obavljaju odvodi na čvoru 3. Aplikacije i odvodi se postavljaju uz pomoć `OnOffHelper` i `PacketSinkHelper` pomoćnika.

```
  PacketSinkHelper sink2 ("ns3::TcpSocketFactory", InetSocketAddress
(interfaces23.GetAddress(1), 10));
  ApplicationContainer apps2 = sink2.Install (allNodes.Get (2));
  apps2.Start (Seconds (1.0));
  apps2.Stop (Seconds (32.0));

  OnOffHelper onOffApp1 ("ns3::TcpSocketFactory", InetSocketAddress
(interfaces23.GetAddress (1), 9));
  onOffApp1.SetAttribute ("DataRate", StringValue ("10Mbps"));
  onOffApp1.SetAttribute ("PacketSize", UintegerValue (2048));
```

```
  onOffApp1.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=30.0]"));
  onOffApp1.SetAttribute ("OffTime", StringValue
("ns3::UniformRandomVariable[Min=1.0|Max=3.0]"));
```

Veličina prozora zagušenja zapisana je u mrežnoj utičnici koju će *on-off* aplikacija koristiti. Za praćenje promjene veličine dodana je funkcija `CwndChange:`

```
void CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
  std::cout << "Congestion window has changed at time "<< Simulator::Now
().GetSeconds () "s from " << oldCwnd << " to " << newCwnd;
}
```

koja promjenu prozora prikazuje na zaslonu. Da bi vrijednosti bile zapisane u datoteku koja će poslije poslužiti za izradu grafova iskorišten je pomoćnik `AsciiTraceHelper`. Njegova metoda `CreateFileStream()` kao argument prima ime datoteke i stvara *stream* tipa `OutputStreamWrapper` koji će biti zapisan u datoteku, a za povrat pruža pokazivač na *stream*.

```
AsciiTraceHelper asciiTraceHelper;
  Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream
("NewReno.txt");
```

Simulacijom je predviđeno da *on-off* aplikacije započinju slanje podataka 2 s nakon početka. Kako u tom trenutku njihove utičnice još nisu stvorene, pokazivači na njih su *nul*-pokazivači. Zbog toga je prije samog pokretanja simulatora, uz pomoć metode `Simulator::Schedule()` zakazuje poziv na funkciju `DoCwndTraceConnectOnSocketAfterAppStarts()` koja je tipa void i prima kao argumente pokazivač na aplikaciju i stream koji ćemo koristit za zapisivanje promjene veličine prozora zagušenja.

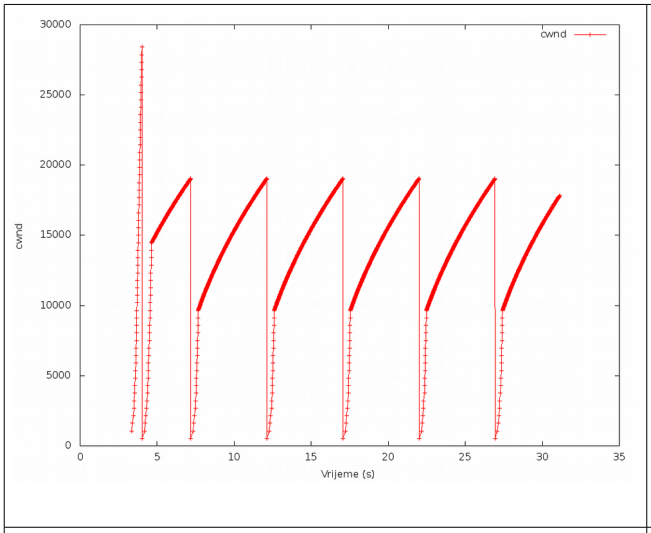
```
void DoCwndTraceConnectOnSocketAfterAppStarts (Ptr<Application> app,
Ptr<OutputStreamWrapper> stream)

{
  NS_ASSERT_MSG (app != 0, "OnOffApplication pointer can't be null");
  Ptr<Socket> appSocket = app->GetObject<OnOffApplication> ()->GetSocket();
  NS_ASSERT_MSG (appSocket != 0, "OnOffApplication socket can't be null, is app
started?");
  appSocket->TraceConnectWithoutContext ("CongestionWindow", MakeBoundCallback
(&CwndChangeWriteFile, stream));
}

int main(){
...
  Simulator::Schedule (Seconds (2.0) + NanoSeconds (1.0),
  &DoCwndTraceConnectOnSocketAfterAppStarts, allNodes.Get (0)->GetApplication
```

```
(0),
  stream);
```

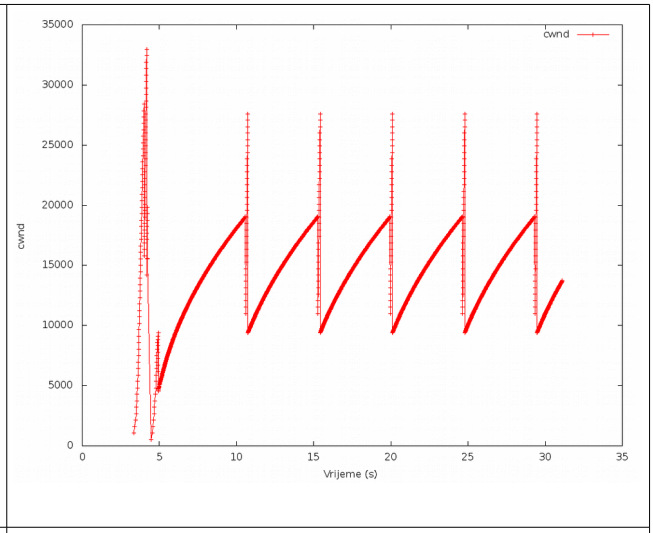
Kao rezultat izvođenja dobivamo tekstualnu datoteku s podacima o veličini prozora zagušenja u određenim vremenskim trenucima koja je potrebna za izradu prikaza su pomoću *gnuplot* aplikacije.

U svim simuliranim situacijama praćenje veličine prozora zagušenja se obavlja samo za *on-off* aplikaciju instaliranu na čvoru N1, a za mjerenje brzine prijenosa uzeti su količina dobivenih podataka i vrijeme trajanja prijenosa između čvorova N1 i N3.

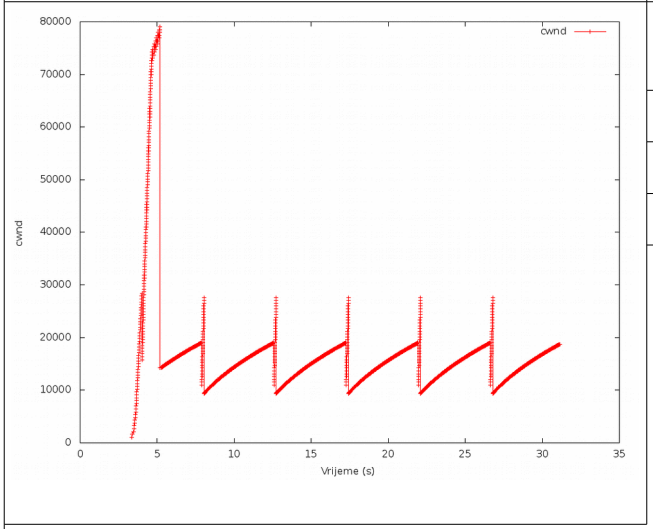
U (Tablica 4) prikazani su rezultati prve simulirane situacije. Simulacija predstavlja stvarnu situaciju u kojoj korisnik na računalu N1 neprekidno 30 s šalje podatke računalu N3 različitim implementacijama TCP-a.



Tcp Tahoe



Tcp Reno



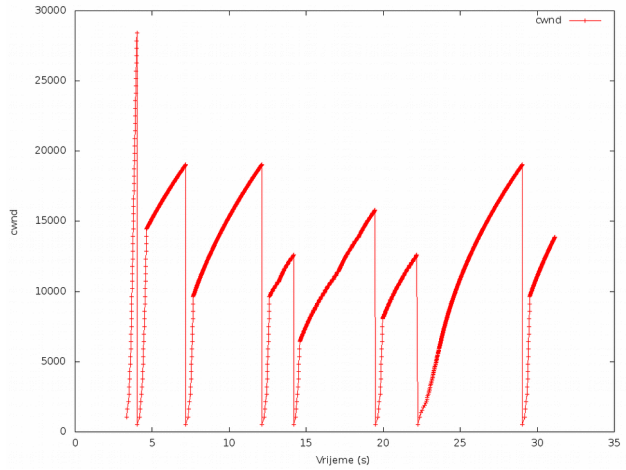
Tcp NewReno

Protokol	Vrijeme (s)	Promet (kB)	Brzina (kbps)
TCP Tahoe	29,1015	2986	820,85
TCP Reno	29,0883	3046	837,73
TCP New Reno	29,0931	3074	845,29

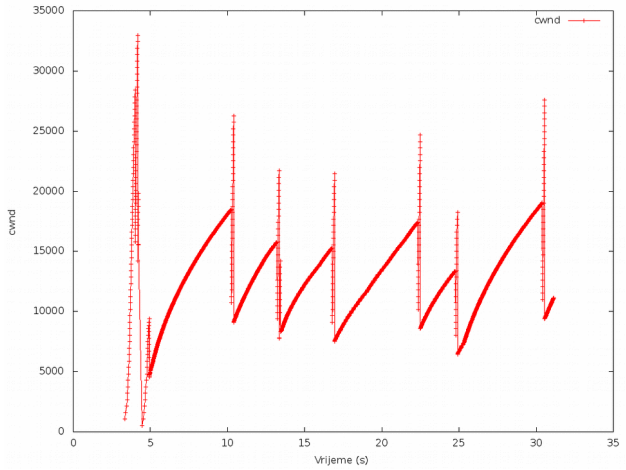
Tablica 4: Prikaz rezultata simulacije 1

Razlike u srednjoj vrijednosti brzine prijenosa pokazuju *Tahoe* gubi značajan dio propusnosti fizičke mreže, najviše zbog smanjenja prozora zagušenja na 1 MSS nakon svakog gubitka segmenta. Prednost koju *New Reno* ima pred implementacijom *Reno* ostvarena je ne izlaskom iz faze brzog oporavka dok svi segmenti poslani do trenutka gubitka segmenta nisu potvrđeni.

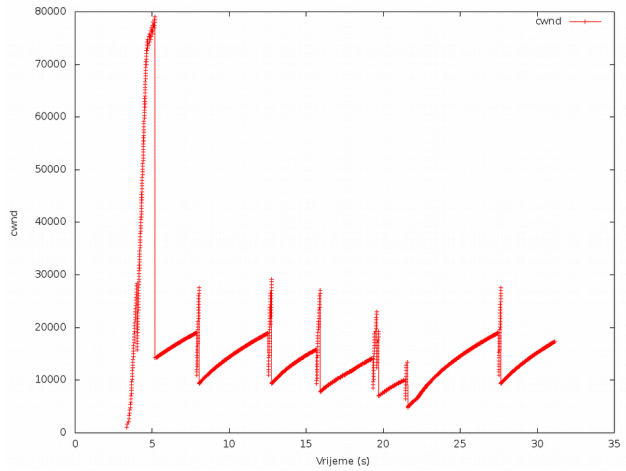
Druga simulacija predstavlja stvarnu situaciju kada 10 s nakon što je korisnik na N1 započeo slanje datoteke, korisnik na računalu N4 također šalje datoteku prema korisniku N3. Rezultati su prikazani u (Tablica 5) .



Tcp Tahoe



Tcp Reno



Tcp NewReno

Protokol	Vrijeme (s)	Promet (kB)	Brzina (kbps)
TCP Tahoe	29,1016	2588	711,44
TCP Reno	29,0997	2710	745,03
TCP New Reno	29,0944	2752	756,71

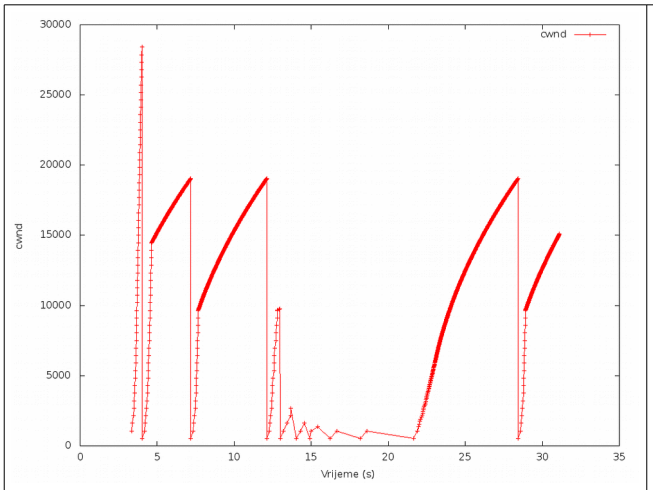
Tablica 5: Prikaz rezultata simulacije 2

Simulacija pokazuje pad srednje brzine prijenosa za sve tri implementacije u odnosu na prvu simulaciju: *Tahoe* je izgubio 13,31 % brzine, dok su *Reno* i *New Reno*

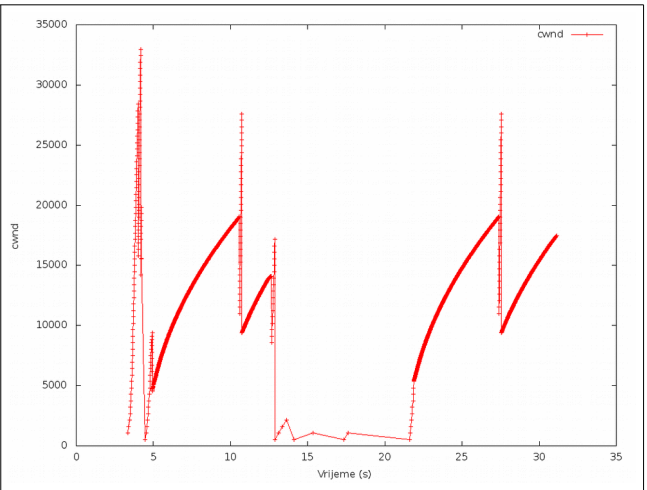
izgubili 11,06% i 10,47%. Gubitci nisu veliki jer računalo N4 također koristi TCP protokol koji pokušava izbjeći zagušenje unutar mreže.

U trećoj simuliranoj situaciji, računalo N4 također 10 s nakon što je N1 započeo slanje podataka, šalje datoteku prema računalu N3 ali koristeći drugi najčešće korišteni protokol transportnog sloja UDP.

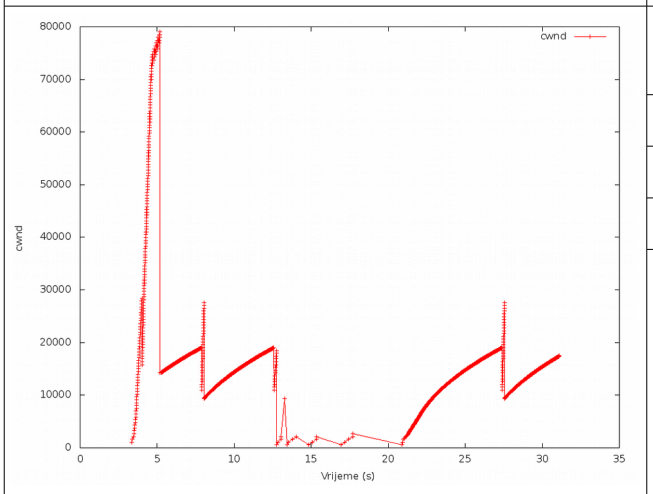
Rezultati su prikazani u (Tablica 6):



Tcp Tahoe



Tcp Reno



Tcp NewReno

Protokol	Vrijeme (s)	Promet (kB)	Brzina (kbps)
TCP Tahoe	29,0928	1924	529,06
TCP Reno	29,0949	2014	553,77
TCP New Reno	29,0922	2082	572,52

Tablica 6: Prikaz rezultata simulacije 3

Protokol UDP ne vodi računa o pouzdanosti prijenosa, već u što kraćem vremenskom razdoblju pokušava dostaviti primatelju što je moguće veću količinu podataka. Rezultat toga je da je UDP u potpunosti zauzeo *bandwith* mreže i sve tri implementacije protokola TCP za vrijeme komunikacije između N4 i N3 nisu ostvarile

prijenos podataka. Pad brzine prijenosa podataka u odnosu na prvu simuliranu situaciju su zbog toga značajni i *Tahoe* je izgubio 33,54 % brzine, dok su *Reno* i *New Reno* izgubili 33,89% i 32,27 %.

4. Zaključak

Veze između udaljenih računala na Internetu mogu prolaziti i preko nekoliko usmjerivača. Porastom broja korisnika i interakcije među njima, svaki od tih usmjerivača je izložen riziku zagušenja što u konačnici rezultira nižom brzinom prijenosa podataka. Rješenje ovog problema je moguće povećanjem propusnosti fizičkog sloja mreže te zamjena usmjerivača koji nemaju mogućnost usmjeravanja dobivene količine paketa, što bi rezultiralo značajnim troškovima. Rezultati simuliranih situacija prikazani u ovom radu pokazuju da je preinakama u algoritmima kontrole zagušenja moguće postići povećanje brzina prijenosa i bez potrebe za promjenom fizičkog sloja. Pri tom, implementacija TCP *New Reno* pokazuje najmanji pad brzine prijenosa podataka pri zagušenju usmjerivača zahvaljujući boljem načinu oporavka nakon gubitka više segmenata iz istog kliznog prozora.

Mrežni simulator *Ns-3* u korištenoj inačici (3.16) pokazuje manja odstupanja modela implementacija od stvarno korištenih. Najveća zamjerka je nedostatak grafičkog sučelja (GUI) prema korisniku, što od korisnika zahtjeva određeno poznavanje objektnog programiranja kao i same programske strukture simulatora. To ga čini neprikladnim za korištenje u edukaciji na nižim razinama obrazovanja. Daljnji razvoj simulatora trebao bi uključiti i druge implementacije protokola. Modularna arhitektura omogućava otklanjanje ovih problema. Za više razine obrazovanja pokazao se kao moćan i fleksibilan alat pri istraživanju mreža.

Literatura

[1] D. D. Clark, “The design Philosophy of the DARPA Internet Protocols,” 1988. [Online]. Available: https://www.cs.princeton.edu/~jrex/teaching/spring2005/reading/clark88.pdf. [Accessed: 23-Sep-2015].

[2] DARPA Internet Program, “RFC 793 Transmission control protocol.” Internet Engineering Task Force (IETF), 1981.

[3] D. E. Comer, *Internetworking with TCP/IP: principles, protocols and architectures*, 4th ed., vol. 1. New Jersey: Prentice Hall, 2000.

[4] R. Stevens, *TCP/IP Illustrated Volume 1:The protocols*, 2nd ed. Michigan: Pearson Education, Inc., 2012.

[5] M. Radovan, *Računalne mreže 2 Prijenos, mrežne usluge i zaštita*. Rijeka: DPT, 2011.

[6] V. Jacobson and M. Karels, “Congestion Avoidance and Control,” 1988. [Online]. Available: https://www.google.hr/search?q=jacobson&ie=utf-8&oe=utf-8&gws_rd=cr&ei=dIrvVcLdO4Wua-H3vbAL#q=jacobson+tcp. [Accessed: 09-Sep-2015].

[7] I. Minić, “Analiza ponašanja transportnog protokola TCP,” 09-Sep-2015. [Online]. Available: https://www.fer.unizg.hr/_download/repository/TCPminic.pdf. [Accessed: 09-Sep-2015].

[8] “IANA-Service Name and Transport Protocol Port Number Registry,” 23-Sep-2015. [Online]. Available: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. [Accessed: 23-Sep-2015].

[9] R. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms - RFC2001.” [Online]. Available: https://tools.ietf.org/html/rfc2001. [Accessed: 24-Sep-2015].

[10] R. Braden, “Requirements for Internet Hosts - Communication Layers - RFC 1122.” [Online]. Available: https://tools.ietf.org/html/rfc1122. [Accessed: 24-Sep-2015].

[11] “RFC 5681 - TCP Congestion Control.” [Online]. Available: https://tools.ietf.org/html/rfc5681. [Accessed: 09-Sep-2015].

[12] R. Chaudhary, S. Sethi, R. Keshari, and S. Goel, “A study of comparison of Network Simulator,” 23-Sep-2015. [Online]. Available: http://www.ijcsit.com/docs/Volume%203/Vol3Issue1/ijcsit2012030129.pdf. [Accessed: 23-Sep-2015].

[13] “Ns-3 Model Library.” [Online]. Available: https://www.nsnam.org/docs/release/3.16/models/singlehtml/index.html. [Accessed: 24-Sep-2015].

[14] “Ns-3 Tutorial.” [Online]. Available: https://www.nsnam.org/docs/release/3.16/tutorial/singlehtml/index.html. [Accessed: 24-Sep-2015].

[15] “ns-3: ns3::PointToPointHelper Class Reference.” [Online]. Available: https://www.nsnam.org/doxygen-release/classns3_1_1_point_to_point_helper.html. [Accessed: 24-Sep-2015].

[16] “ns-3: ns3::DropTailQueue Class Reference.” [Online]. Available: https://www.nsnam.org/doxygen-release/classns3_1_1_drop_tail_queue.html. [Accessed: 24-Sep-2015].

[17] “ns-3: ns3::InternetStackHelper Class Reference.” [Online]. Available: https://www.nsnam.org/doxygen-release/classns3_1_1_internet_stack_helper.html. [Accessed: 24-Sep-2015].

[18] “ns-3: ns3::Ipv4AddressHelper Class Reference.” [Online]. Available: https://www.nsnam.org/doxygen-release/classns3_1_1_ipv4_address_helper.html. [Accessed: 24-Sep-2015].