- Protokol transportnog sloja, obezbedjuje **logicku komunikaciju** izmedju **procesa** aplikacija koje se izvrsavaju na razlicitim racunarima. Pod logickom komunikacijom misli se na to da sa stanovista aplikacije izgleda kao da su racunari na kojima se procesi izvrsavaju neposredno povezani.
- Protokoli transportnog sloja ostvaruju se u **krajnjim sistemima**, a ne u mreznim ruterima.
- Na predajnoj strani, transportni sloj pretvara poruku koju dobije od procesa aplikacije posiljaoca u pakete transportnog sloja, koji se nazivaju **segmenti** transportnog sloja. Transportni sloj zatim prenosi te segmente do mreznog sloja predajnog kranjeg sistema, gde se ti segmenti ekapsuliraju u pakete mreznog sloja (datagrame) i salju na odrediste.
- Mrezni ruteri obradjuju samo polja datagrama mreznog sloja tj ne ispituju polja segmenta transportnog sloja koja su enkapsulirana unutar datagrama.
- Protokol mreznog sloja (kao npr **IP**), obezbedjuje logicku komunikaciju izmedju **racunara**.
- Transportni protokol ni na koji nacin ne utice na nacin prenosenja poruke unutar same mreze.
- Usluge koje transportni protokol moze da obezbedi obicno su ogranicene modelom usluga protokola mreznog sloja koji se nalazi ispod njega.
- Transportni protokol moze da ponudi aplikaciji uslugu pouzdanog prenosa podataka cak i kada mrezni protokol nije narocito pouzdan, tj cak i kada gubi, ostecuje, duplira pakete.
- Transportni protokol moze koristiti **sifrovanje** kojim se garantuje da poruke aplikacija nece citati neovlascena lica, cak i ako mrezni sloj ne moze da garantuje poverljivost za segmente transportnog sloja.
- Model usluga koji nudi protokol IP je **usluga najboljeg pokusaja isporuke,** tj IP cini sve sto moze da isporuci segmente izmedju racunara, *ali ne daje nikakvu garanciju*, ne garantuje isporuku segmenta, redosled isporuke segmenata i ne garantuje integritet podataka u segmentima. Zato se za protokol IP kaze da je **nepouzdana usluga**.
- Najvazniji zadatak protokola UDP i TCP jeste da izvrse uslugu isporuke izmedju dva **procesa**. Prosirivanje isporuke od racunara do racunara na isporuku od procesa do procesa, naziva se **multipleksiranje** i **demultipleksiranje transportnog sloja.**
- TCP protokol nudi **pouzdan prenos podataka** i obezbedjuje **kontrolu zagusenja.** Nastoji da svim TCP vezama koje prelaze preko zagusenog mreznog linka dodeli jednak deo propusnog opsega.

Multipleksiranje i demultipleksiranje

- Svaki soket ima jedinstven identifikator.
- Svi segmenti transportnog sloja imaju skup polja. Transportni sloj ispituje ova polja da bi odredio prijemni soket i zatim usmerava segment u taj soket.
- Ovaj zadatak isporucivanja podataka iz segmenta transportnog sloja u sodgovarajuci soket, naziva se **demultipleksiranje**.
- Prikupljanje delova podataka, enkapsuliranje svakog dela dodavanjem zaglavlja, da bi se napravili segmenti i predavanje tih segmenata mreznom sloju, naziva se **multipleksiranje**.

- Za multipleksioranje, neophodno je da soketi imaju jedinstvene identifikatore i da svaki segment sadrzi posebna polja koja naznacavaju soket u koji odredjeni segment treba isporuciti. Ta polja su: **polje broja izbornog porta** i **polje broja odredisnog porta.**
- Broj porta je 16-bitni broj u rasponu od 0 do 65535. Brojevi portova od 0 do 1023 se nazivaju **dobro poznatim brojevima portova**, i rezervisani su za protokole opste poznatih aplikacija.
- UDP soket je potpuno odredjen dvodelnim podatkom koji se sastoji od **odredisne IP adrese** i **broja odredisnog porta**.
- TCP soket je potpuno odredjen cetvrodelnom oznakom koju cine: **izvorna IP adresa**, **broj izvornog porta**, **odredisna IP adresa** i **broj odredisnog porta**. (zahtev za uspostavljanje TCP veze je TCP segment sa brojem odredisnog porta i posebnim bitom za uspostavljanje veze koji je postavljen u TCP zaglavlju)
- Serverski racunar moze da podrzi vise istovremenih TCP soketa, pri cemu su svi soketi pridruzeni nekom procesu, a svaki soket prepoznaje se po sopstvenoj cetvorodelnoj oznaci.
- Broj soketa nije uvek jednak broju procesa.

Prenos bez uspostavljanja veze: protokol UDP

- Aplikacije koriste UDP protokol iz razloga:
  1) Bolja kontrola na nivou aplikacije toga sta se salje i kada se salje.
  2) Sema uspostavljanja veze (nema kasnjenja zbog iste).
  3) Nema stanja veze.
  4) Malo dodatno zaglavlje paketa (UDP ima samo 8 dodatnih bajtova, dok TCP ima 20)
- Neki od protokola koji koriste UDP: **SNMP** – upravljanje mrezom. **RIP** – protokol za rutiranje. **DNS** – prevodjenje naziva racunara u IP adrese.
- Jeste moguce da aplikacija dobije pouzdan prenos podataka i kada koristi UDP, ako se pouzdanost ugradi u samu aplikaciju.
- UDP zaglavlje ima samo 4 polja od kojih svako zauzima po 2 bajta, a to su:
  1. Broj izvornog porta.
  2. Broj odredisnog porta.
  3. Duzina.
  4. Kontrolni zbir.
- Kontrolni zbir se koristi kako bi se utvrdilo da li su bitovi unutar UDP segmenta promenjeni prilikom prenosa od izvora do odredista, tako sto UDP na strani posiljaoca izracunava komplement jedinice za sumu svih 16-bitnih reci u segmentu. Ovaj rezultat se stavlja u polje kontrolnog zbira UDP segmenta.
- UDP koristi ovaj sistem jer ne postoji garancija da svi linkovi od izvora do odredista obezbedjuju proveru gresaka i zato sto mora da obezbedi otkrivanje gresaka na transportnom sloju, *od jednog do drugog kraja*.
- UDP nista ne preduzima da bi se greska ispravila.

Principi pouzdanog prenosa podataka

- Protokol pouzdanog prenosa podataka ima zadatak da obezbedi da se bitovi koji se prenose nepouzdanim kanalom ne ostecuju, ne gube i da se svi isporucuju redosledom po kome su poslati.
- TCP protokol ostvaruje prenos podataka preko nepouzdanog (IP) mreznog sloja.
- Predajna strana protokola pokrece se odozgo, pozivom *rdt_send()* (*rdt* – **reliable** data transfer, *_send* – znaci da se poziva predajna strana protokola *rdt*). Kada paket pritigne iz prijemne strane kanala, na prijemnoj strani se poziva *rdt_rcv()*. Kada protokol *rdt* zeli da isporuci primljene podatke gornjem sloju, to radi pozivom *deliver_data*.
- Osim razmene paketa koji sadrze podatke koje treba preneti, predajna i prijemna strana protokola *rdt* moraju takodje da razmenjuju i **kontrolne pakete** u oba smera. Obe strane protokola salju pakete drugoj strani pozivanjem *udt_send*() (*udt* – **unreliable** transfer – nepouzdan transfer podataka).

- Pouzdan prenos podataka preko savrseno pouzdanog kanala: protokol rdt1.0

- Sam protokol rdt1.0 je jednostavan.
- FSM – masine konacnog stanja.
- Za primaoca i posiljaoca postoje **zasebne** masine konacnog stanja, koje imaju **samo po jedno stanje.**
- Predajna strana protokola *rdt*, prihvata pordatke iz gornjeg sloja dogadjajem *rdt_send(data)*, pravi paket koji sadrzi te podatke (postupkom *make_pkt(data)*) i salje paket u kanal. U praksi, dogadjaj *rdt_send(data)* nastaje tako sto aplikacija iz gornjeg sloja poziva neku proceduru (recimo, proceduru *rdt_send()*).
- Na prijemnoj strani, *rdt* prima paket iz kanala ispod sebe dogadjajem *rdt_rcv(data)*, izvlaci podatke iz paketa (postupkom *extract(packet,data)*) i prenosi podatke gornjem sloju (postupkom *deliver_data(data)*). U praksi, dogadjaj *rdt_rcv(paket)* nastaje tako sto protokol nizeg sloja poziva odgovarajucu proceduru (recimo,proceduru *rdt_rcv()*).

- Pouzdan prenos podataka preko kanala sa bitskim greskama: protokol rdt2.0

- Ovaj protokol se sastoji od **pozitivnih poruka(ACK)** i **negativnih poruka(NAK)**, kojima se saopstava sta je primljeno pravilno, a sta treba poslati ponovo.
- U racunarskim mrezama, protokoli za transfer podataka koji se zasnivaju na ponovnom slanju, poznati su kao **ARQ**(Automatic Repeat reQuest) protokoli.
- Da bi se izborili sa bitskim greskama, potrebno je ARQ protokole dopuniti sa jos tri dodatne mogucnosti:

  1.Otkrivanje gresaka (potrebno je da posiljalac posalje primaocu dodatne bitove koji se smestaju u polje kontrolnog zbira paketa podataka protokola rdt2.0)

  2. Povratna informacija od primaoca posiljaocu. (pozitivne(ACK(i negativne(NAK) povratne informacije; ti paketi sadrze samo jedan bit, 1-ACK, 0-NAK)

  3. Ponovno slanje.
- Predajna strana protokola rdt2.0 ima **dva stanja.**

- Posiljalac u stanju cekanja na ACK ili NAK paket, ne moze da preuzima druge podatke sa gornjeg sloja i da ih salje. Zbog toga se naziva – protokol **stani i cekaj**.
- Dogadjaj oznacen sa *rdt_rcv(rcvpkt) && isACK(rcvpkt)* znaci da je poslednji paket ispravno primljen.
- Dogadjaj oznacen sa *rdt_rcv(rcvpkt) && corrupt(rcvpkt)* znaci da je utvrdjeno da paket sadrzi gresku.
- Pri koriscenju ACK i NAK paketa, moze doci i do njihovog ostecenja. Tada posiljalac jednostavno ponovo posalje paket podataka, pri cemu moze doci do dupliranja paketa. Ovaj problem se resava dodavanjem **rednog broja** za svaki paket, tako da primalac samo treba da proveri da li primljeni paket predstavlja ponavljanje.
- Poboljsana verzija protokola rdt2.0 je protokol rdt2.1, koji ima **dva puta vise stanja nego ranije.**
- Protokol **rdt2.1** koristi **i negativne i pozitivne potvrde** od primaoca ka posiljaocu.
- Protokol **rdt2.2** koristi **samo pozitivne potvrde.** Umesto negativne potvrde, dovoljno je poslati dve ACK potvrde i posiljalac ce znati da primalac nije ispravno primio paket koji sledi *iza* paketa za koji su poslate dve ACK potvrde. Za ovaj protokol, primalac takodje mora da dodeli redni broj paketa ciji prijem potvrduje ACK porukom (to se postize dodavanjem vrednosti ACK – 0 ili ACK – 1 u *make_pkt()*.

<u>- Pouzdan prenos podataka preko kanala sa bitskim greskama i gubicima paketa: protokol rdt3.0</u>

- Za otkrivanje i resavanje problema izgubljenih paketa zaduzuje se **posiljalac**.
- Ukoliko se izgubi paket, potrebno je poslati ga ponovo. Za primenu mehanizma ponovnog slanja na osnovu vremena, potreban je **tajmer** koji moze da posalje prekid posiljaocu po isteku zadatog vremena. Posiljalac ce morati da bude u stanju da pokrene tajmer *uvek* kada posalje paket, da odgovori na prekid od strane tajmera i da zaustavi tajmer.
- Posto redni brojevi paketa naizmenicno menjaju vrednost izmedju 0 i 1, protokol rdt3.0 se naziva i **protokol naizmenicnih bitova**.

- <u>Elementi neophodni za rad protokola su kontrolni zbirovi, redni brojevi, tajmeri i pozitivne i negativne potvrde prijema paketa.</u>

<u>- Pouzdani cevovodni protokol za prenos podataka</u>

- Kljuc problema performansi protokola rdt3.0 je sto je to protokol sa stajanjem i cekanjem.
- Resenje je jednostavno: umesto da radi tako sto stoji i ceka, posiljaocu se dozvoljava da salje vise paketa ne cekajuci potvrde prijema. Posto prolazak vise paketa izmedju posiljaoca i primaoca moze da se zamisli kao punjenje cevovoda, ova tehnika se naziva **cevovodnom obradom**.
- Cevovodna obrada ima i odredjene posledice na protokole za pouzdan prenos podataka.

1) Mora se povecati raspon rednih brojeva.

2) Predajna i prijemna strana moraju da imaju memoriju za privremeno cuvanje vise paketa – bafer.

3) Potreban raspon rednih brojeva i velicine privremene memorije zavisice od toga kako protokol za prenos podataka postupa u slucaju izgubljenih i ostecenih paketa.

- Dva osnovna pristupa za oporavak od gresaka cevovodne obrade: **Vrati-za-N (Go-Back-N)** tj **GBN**, i **selektivno ponavljanje.**

- Protokol GBN

- Protokol GBN dozvoljava posiljaocu da posalje vise paketa bez cekanja na povtrdu, ali ogranicava najveci dozvoljeni broj *N* nepotvrdjenih paketa u cevovodu.
- *base* – redni broj najstarijeg nepotvrdjenog paketa
- *nextseqnum* – najmanji neupotrebljeni redni broj tj redni broj sledeceg paketa koji treba poslati
- Redni brojevi u intervalu:

  -[0, *base*-1] – odgovaraju paketima koji su vec preneti i ciji je prijem potvrdjen,

  -[*base, nextseqnum*-1] – odgovaraju paketima koji su poslati, ali ciji prijem jos nije potvrdjen,

  -[*nextseqnum, base*+*N*-1] – odgovaraju paketima koji mogu da se posalju odmah cim podaci stignu iz gornjeg sloja,

  -redni brojevi veci ili jednaki od *base+N* ne mogu se koristiti sve dok ne stigne potvrda prijema za nepotvrdjeni paket koji se trenutno nalazi u cevovodu
- Za *N* se kaze da je **velicina prozora**.
- Za protokol GBN se kaze da je **protokol sa kliznim prozorom**.
- GBN posiljalac mora da odgovori na 3 vrste dogadjaja:

  1. Poziv odozgo (kada se odozgo pozove *rdt_send()*, posiljalac proverava da li je prozor pun, tj da li postoji *N* nepotvrdjenih paketa; ako prozor nije pun, paket se salje, ako jeste, paket se vraca gornjem sloju)

  2.Prijem ACK poruke (u ovom protokolu, potvrda prijema za paket sa rednim brojem *n*, smatra se **kumulativnom potvrdom prijema**, koja oznacava da je primalac pravilno primio sve pakete sa rednim brojevima koji *su manji i jednaki n*)

  3. Istek vremena tajmera (posle isteka odredjenog vremena, posiljalac ponovo salje **sve** pakete koji su prethodno poslati, a ciji prijem nije jos uvek potvrdjen)
- U ovakvom **programiranju zasnovanom na dogadjajima**, razlicite procedure se pozivaju bilo iz drugih procedura u skupu protokola ili kao rezultat prekida.

- Selektivno ponavljanje

- Protokoli sa selektivnim ponavljanjem(selective repeat) izbegavaju nepotrebna ponovna slanja tako sto posiljalac ponovo *salje samo one pakete za koje sumnja da ih je primalac pogresno primio.*

- Ovakvo pojedinacno ponovno slanje prema potrebi zahteva da primalac **zasebno** potvrdjuje ispravno primljene pakete.
- SR primalac se potvrditi ispravno primljeni paket bez obzira na to da li je stigao po redu. Paketi izvan redosleda cuvaju se u privremenoj memoriji dok ne stignu nedostajuci paketi.
- Za protokol SR vazi da se prozori posiljaoca i primaoca ne poklapaju uvek.

Transport sa uspostavljanjem veze: protokol TCP

- Za protkol TCP se kaze da je **sa uspostavljanjem veze** zato sto pre pocetka slanja podataka, dva procesa moraju prvo „da se rukuju" tj moraju jedan drugom da posalju neke uvodne segmente.
- Protokol TCP se izvrsava samo u **krajnjim sistemima**, a ne na usputnim elementima mreze, pa oni ne odrzavaju stanje TCP veze, cak nisu uopste svesni postojanja TCP veze.
- TCP veza obezbedjuje **punu dupleksnu uslugu** (A ka B, i B ka A), i to je veza **od tacke do tacke** (tacno izmedju jednog posiljaoca i jednog primaoca). Nije moguc prenos podataka od jednog posiljaoca ka vise primalaca u istoj operaciji slanja.
- Posto se medju racunarima, pri usaglasavanju, razmenjuju tri segmenta, ovaj postupak se cesto naziva **trostruko usaglasavanje**.
- Klijentski proces propusta niz podataka kroz soket. TCP usmerava ove podatke u **predajnu privremenu memoriju** te veze. S vremena na vreme, TCP zahvata delove podataka iz predajne privremene memorije.
- Najveca kolicina podataka koja moze da se zahvati i stavi u jedan segment, ogranicena je **najvecom velicinom segmenta** (Maximum Segment Size) tj **MSS**. To je maksimalna kolicina podataka *aplikativnog sloja* u segmentu.
- MSS se obicno postavlja tako sto se prvo odredi duzina najveceg okvira sloja veze koji moze da posalje lokalni racunar posiljalac, takozvani **MTU** (Maximum Transmision Unit) – **najveca jedinica prenosa**, i zatim postavljanjem vrednosti MSS tako da se osigura da TCP segment moze da stane u jedan okvir sloja veze. Uobicajene MTU vrednosti su 1460 bajtova, 536 bajtova i 512 bajtova.
- TCP dopunava svaku celinu podataka klijenta TCP zaglavljem i tako pravi **TCP segmente.**
- TCP veza se sastoji od privremene memorije, promenljivih i soketa veze sa procesom na jednom racunara, i jos jednog skupa od privremene memorije, promenljivih i soketa veze sa procesom na drugom racunaru.

-Struktura TCP segmenta

- TCP segment se sastoji od vise polja zaglavlja i jednog polja podataka. Polje podataka sadrzi komad podataka aplikacije.
- Zaglavlje TCP segmenta sadrzi:
  -**broj izvornog i odredisnog porta**
  -**polje kontrolnog zbira**
  -32-bitno **polje rednog broja**
  -32-bitno **polje broja potvrde**
  16-bitno **polje prijemnog prozora** (koristi se za kontrolu toka)
  -4-bitno **polje duzine zaglavlja**
  -neobavezno **polje opcija** promenljive duzine (koristi se kada posiljalac i primalac pregovaraju o maksimalnoj duzini segmenta MSS)
  -**polje oznaka** sadrzi 6 bitova:
    ACK,
    RST, SYN i FIN – koriste se prilikom uspostavljanja i prekidanja veze
    PSH – znaci da primalac treba odmah da prosledi podatke gornjem sloju
    URG – koristi se da bi oznacio „hitne" segmente

- Dva najvaznija polja u zaglavlju TCP segmenta jesu redni broj i broj potvrde.
- Redni brojevi odnose se na neprekidni tok prenetih bajtova, a *ne* na niz prenetih segmenata. **Redni broj za segme**nt je redni broj prvog bajta u segmentu unutar neprekidnog toka bajtova.
- *Broj potvrde koji racunar A stavlja u svoj segment je redni broj sledeceg bajta koji racunar A ocekuje od racunara B.*
- Posto TCP potvrdjuje samo bajtove od prvog nedostajuceg bajta u neprekidnom toku, za TCP se kaze da daje **kumulativne potvrde prijema.**

-Procena vremena povratnog puta i isteka vremena tajmera

- Protokol TCP, za oporavak od izgubljenih segmenata koristi mehanizam isteka vremena tajmera i ponavljanje slanja.
- Uzorak vremena povratnog puta RTT za neki segment, koji se naziva *SampleRTT*, jeste vreme od trenutka slanja segmenta, do prijema potvrde tog segmenta. U vecini TCP protokola, meri se samo jedan uzorak *SampleRTT*, tj meri se za samo jedan preneti, ali jos uvek nepotvrdeni segment.
- TCP stalno izracunava prosek izmerenih vrednosti *SampleRTT*, nazvan *EstimatedRTT*, prema obrascu: $EstimatedRTT = (1- \alpha)*EstimatedRTT + \alpha * SampleRTT$ (α=0.125)
- *EstimatedRTT* je ponderisani prosek vrednosti *SampleRTT.* Ovakav prosek se naziva **eksponencijalno ponderisani klizni prosek,** zato sto znacaj (ponder) date verdnosti *SampleRTT* eksponencijalno opada sa svakim sledecim azuriranjem.
- Varijacija vremena povratnog puta – *DevRTT*, predstavlja procenu standardne devijacije *SampleRTT* od *EstimatedRTT:*

  $DevRTT = (1 - \beta)*DevRTT + \beta*(SampleRTT - EstimatedRTT)$ ($\beta = 0.25$)
- Vreme koje bi trebalo da protekne pre ponovnog slanja bi trebalo da bude jednako *EstimatedRTT* uz neku rezervu. To vreme se odredjuje na sledeci nacin:

  $TimeoutInterval = EstimatedRTT + 4*DevRTT$

- TCP je protokol sa samo jednim tajmerom.
- Kad god nastupi dogadjaj isteka vremena, TCP ponovo salje nepotvrdjeni segment sa najmanjim rednim brojem. Ali, svaki put kada TCP ponovo salje neki segment, on **udvostrucava vremen trajanja tajmera** u odnosu na prethodnu vrednost umesto da ga onovo izracuna.
- Medjutim, ako se tajmer ponovo pokrece posto se, ili prime podaci iz aplikacije odozgo, ili se primi ACK potvrda, vrednost *TimeoutInterval* izracunava se od najnovijih vrednosti *EStimatedRTT* i *DevRTT,* cime se postize kontrola zagusenja u ogranicenoj meri.
- Posiljalac cesto moze da otkrije gubljenje paketa mnogo pre isteka ovog vremena kada primi takozvane ponovljene ACK potvrde. **Ponovljeni ACK** je ACK kojim se ponovo potvrdjuje prijem segmenta za koji je posiljalac vec ranije primio takvu potvrdu.
- Kada TCP primalac primi segment sa rednim brojem vecim od sledeceg ocekivanog rednog broja, on primecuje prazninu u toku podataka, tj primecuje da neki segment nedostaje. Posto TCP ne koristi negativne potvrde prijema, on jednostavno ponovo potvrdjuje tj pravi jos jednu ACK potvrdu.
- U slucaju kada primi tri ponovljene ACK potvrde, TCP posiljalac preduzima **brzo ponovno slanje**, ponovo saljuci nedostajuci semgent *pre* isteka vremena tajmera.
- TCP – mesavina GBN protokola i protokola sa selektivnim ponavljanjem.

-Kontrola toka

- Kada TCP veza primi bajtove koji su ispravni i u pravilnom redosledu, ona ih stavlja u prijemnu privremenu memoriju. Odgovarajuci proces aplikacije cita podatke iz ove privremene memroije, ali ne obavezno cim ti podaci stignu. Ako aplikacija relativno sporto ucitava podatke, posiljalac moze vrlo lako da preplavi prijemnu privremenu memoriju.
- TCP nudi **uslugu kontrole toka**, kojom se uskladjuje brzina kojom posiljalac salje podatke, sa brzinom kojom prijemna aplikacija ucitava podatke.
- TCP obezbedjuje kontrolu toka tako sto *posiljalac* odrzava promenljivu nazvanu **prijemni prozor**, kako bi mogao da nasluti koliko ima mesta u privremenoj memoriji primaoca. Posto TCP radi u punom dupleksu, posiljaoci na obe strane veze odrzavaju *zaseban prijemni prozor*.
- *LastByteRead*: broj poslednjeg bajta u toku podataka koji je proces aplikacije u racuanru B procitao iz privremene memorije
- *LastByteRcvd*: broj poslednjeg bajta u toku podataka koji je stigao sa mreze i smesten u prijemnu privremenu memoriju racunara B
- Posto TCP ne sme da prepuni dodeljenu privpremenu memoriju, uvek mora da bude: $LastByteRcvd - LastByteRead \leq RcvBuffer$
- Prijemni prozor, oznacen sa *RcvWindow,* postavljen je na velicinu slobodnog prostora u privremenoj memoriji: $RcvWindow = RcwBuffer - [LastByteRcvd - LastByteRead]$
- *RcvWindow* **nema stalnu vrednost.**
- Racunar A, tokom citavog trajanja veze obezbedjuje da je $LastByteSent - LastByteAcked \leq RcvWindow$
- Ukoliko se racunaru A, objavi da je $RcvWindow = 0$, TCP zahteva da racunar A nastavi da salje segmente sa po jednim bajtom podataka kada velicina prijemnog prozora

racunara B bude jednaka nuli. U jednom trenutku, privremena memorija pocinje da se prazni i potvrde prijema prenose racunaru A vrednost *RcvWindow* razlicitu od nule.

- Tokom trajanja TCP veze, protokol TCP koji se izvrsava na oba racuanra, prolazi kroz razlicita **TCP stanja.** Prekidanje TCP veze se izvrsava u 4 koraka.
- Klasican napad DDoS je **napad plavljenjem SYN segmentima**. Uspesna odbrana je nazvana **SYN kolacici**.
- Ukoliko racunar primi TCP segment ciji se broj porta ili izvorna IP adresa ne poklapaju ni sa jednim postojecim soketom na racuanru, taj racunar ce tada poslati izvornom racuanaru poseban segment za ponistavanje zahteva. Ovaj TCP segment sadrzi bit **RST** postavljen na 1. Kada racunar primi UDP paket ciji se broj odredisnog porta ne poklapa ni sa jednim od pokrenutih UDP soketa, on salje poseban ICMP datagram.

Principi kontrole zagusenja

- Posledice zagusene mreze:

1.Veliko kasnjenje u redovima za cekanje koje nastaje kad se brzina pristizanja paketa priblizava kapacitetu linka.

2.Posiljalac mora ponovo da salje podatke da bi nadoknadio pakete koji su odbaceni zbog prepunjene privremene memorije.

3.Nepotrebno ponovno slanje izazvano velikim kasnjenjem dovodi do toga da ruter trosi propusni opseg linka za prosledjivanje nepotrebnih kopija paketa.

4.Kada se paket odbaci negde na putanji, to znaci da je prenosni pakacitet svih linkova upotrebljenih za prosledjivanje paketa do trenutka odbacivanja paketa uzaludno utrosen.

- Brzina kojom transportni sloj salje segmente koji sadrze prvobtine podatke *kao i* podatke koji se ponovo salju, ponekad se naziva **ponudjeno opterecenje mreze**.

-Resenja koja se koriste za kontrolu zagusenja

- **Kontrola zagusenja sa kraja na kraj** – mrezni sloj ni na koji nacin *ne pomaze* transportnom sloju sto se tice kontrole zagusenja.
- **Kontrola zagusenja uz pomoc mreze** – komponente mreze tj ruteri daju posiljaocu jasne povratne informacije o stanju zagusenja u mrezi. Ovu povratnu informaciju moze da sacinjava samo jedan bit koji ukazuje na zagusenje linka. Informacija o zagusenju se prosledjuje od mreze ka posiljaocu na jedan od dva nacina:

1)Neposredna povratna informacija se sa mreznog rutera salje posiljaocu. Ova vrsta obavestenja je obicno u obliku paketa zagusenja.

2)Ruter oznacava odgovarajuce polje u paketu koji putuje od posiljaoca ka primaocu kako bi ukazao na zagusenje. Kada primi tako oznaceni paket, primalac obavestava posiljaoca da je dobio upozorenje o zagusenju.

- **ABR** – algoritam za kontrolu zagusenja u ATM mrezama – mreze koje za komutiranje paketa koriste virtuelna kola(VC), sto znaci da svaki komutator na putanji od izvora do odredista vodi racuna o stanju odredjenog virtuelnog kola od izvora do odredista.
- Kada je mreza manje opterecena, ABR usluga moze slobodno da koristi dostupan visak propusnog opsega. Kada je mreza zagusena, ABR usluga bi trebalo da prigusi brzinu prenosa na neku unapred postavljenu najmanju brzinu.
- U ABR usluzi, celije podataka prenose se od izvora do odredista kroz niz usputnih komutatora. Izmedju celija podataka umetnute su **celije za upravljanje resursima**, **celije RM**. Ove RM celije mogu se upotrebiti za razmenjivanje informacija o zagusenju medju racunarima i komutatorima.
- ABR kontrola zagusenja u ATM mrezi koristi pristup koji se zasniva na **brzini**.
- ABR predvidja 3 mehanizma kojima komutatori upozoravaju primaoce o zagusenju u mrezi:

1.**EFCI bit**: Svaka *celija podataka* sadrzi **bit za nedvosmisleno upozorenje o zagusenju unapred tj bit EFCI**. Zaguseni mrezni komutator moze da postavi EFCI bit u celiji podataka na 1 i tako upozori odredisni racunar na zagusenje. Tada odredisni racunar postavlja bit za upozorenje o zagusenju, bit CI u RM celiji na 1, i RM celiju salje nazad posiljaocu.

2. **Bitovi CI i NI**: Broj umetnutih RM celija moze da se podesava parametrom cija je podrazumevana vrednost jedna RM celija na svakih 32 celije podataka. Te RM celije sadrze bit za upozorenje o zagusenju (CI bit) i **bit „ne povecavaj", bit NI** (no increase). Komutator moze da unutar RM celije koja prolazi krzo njega u slucaju blagog zagusenja psotavi bit NI na 1, a u slucaju velikog zagusenja bit CI postavi na 1.

3. **Postavljanje vrednosti ER**: Svaka RM celija takodje sadrzi dvobajtno **polje eksplicitno zadate brzine, polje ER**. Zaguseni komutator moze da smanji vrednost u polju ER unutar RM celije koja prolazi kroz njega.

- ABR izvor ATM mreze prilagodjava brzinu kojom salje celije u zavisnosti od vrednosti u poljima CI, NI i ER u vracenoj RM celiji.

TCP kontrola zagusenja

- TCP mehanizam za kontrolu zagusenja zahteva da se sa obe strane veze odrzava jos jedna promenljiva, prozor zagusenja, oznacacen sa *CongWin* (congestion window), koji namece ogranicenje brzine kojom posiljalac moze da salje saobracaj u mrezu.
- Kolicina nepotvrdjenih podataka kod posiljaoca ne sme da predje manju od vrednosti *CongWin* i *RcvWindow* tj : *LastByteSent – LastByteAcked* ≤ min{*CongWin, RcvWindow*}
- *Brzina slanja posiljaoca iznosi priblizno CongWin*/RTT *bajtova u sekundi. Prema tome, posiljalac moze da podesi brzinu slanja podataka u vezu podesavanjem vrednosti CongWin.*

- Ako potvrde stizu relativno sporto, tada se prozor zagusenja relativno sporo povecava. Ako potvrde stizu brzo, prozor zagusenja se povecava mnogo brze.
- Posto TCP koristi potvrde prijema kako bi pokrenuo postupak povecanja prozora zagusenja, za TCP protokol se kaze da ima **vlastiti ritam**.
- TCP protokol za kontrolu zagusenja ima 3 osnovna dela:
  1. **Aditivno povecanje, multiplikativno smanjenje**. Osnovna zamisao na kojoj se zasniva TCP kontrola zagusenja jeste da posiljalac smanji brzinu slanja kada dodje do gubitka. TCP koristi tzv pristup „multiplikativnog smanjenja" tako sto **prepolovi** trenutnu vrednost *CongWin* svaki put kada nastupi dogadjaj gubitka. Vrednost *CongWin* moze i dalje da pada, ali ne ispod 1 MSS.
     Ako nije uoceno zagusenje, postoji dostupna propusna moc koju TCP posiljalac iskoriscava povecanjem vrednosti *CongWin* po malo svaki put kada primi potvrdu prijema. Uobicajeno je da TCP posiljalac poveca *CongWin* za MSS/*CongWin* bajtova kada primi novu potvrdu, sto predstavlja „aditivno povecanje".
     Faza linearnog povecanja brzine pri kontroli zagusenja u protokolu TCP poznata je kao **izbegavanje zagusenja.** Vrednost *CongWin* stalno prolazi kroz cikluse tokom kojih se linearno povecava, a zatim naglo smanjuje na polovinu trenutne vrednosti.
  2. **Spori start.** Na pocetku TCP veze, vredsnot *CongWin* obicno se postavlja na jedan MSS, sto daje pocetnu brzinu slanja priblizno MSS/RTT. U toj **pocetnoj fazi**, umesto linearnog povecanja brzine, TCP posiljalac povecava brzinu eksponencijalno tako sto za svaki RTT **udvostrucava** vrednost *CongWin.* Posiljalac pocinje prenos malom brzinom, ali eksponencijalno povecava brzinu slanja. Eksponencijalni rast se postize tako sto uvecava vrednost *CongWin* za 1 MSS za svaku primljenu potvrdu za poslate segmente.
  3. **Postupak u slucaju isteka vremena tajmera.**
     Ponasanje TCPa u zavisnosti od dogadjaja gubitka:
     1**)**Kada primi trostruku ACK potvrdu, TCP se ponasa tako da se prozor zagusenja prepolovi, a zatim se linearno povecava.
     2)Ako nastupi dogadjaj istek vremena tajmera, TCP posiljalac ulazi u fazu sporog starta tj podesava prozor zagusenja na 1 MSS, a zatim ga eksponencijalno povecava. Prozor i dalje eksponencijalno raste sve dok *CongWin* ne dostigne polovinu vrednosti koju je imao pre dogadjaja isteka vremena tajmera. Od tog trenutkam *CongWin* raste linearno, kao sto se inace ponasa nakon trostruke ACK potvrde.

     TCP koristi promenljivu *Threshold* koja odredjuje velicinu prozora pri kojoj se prekida faza sporti start i pocinje faza izbegavanje zagusenja. Kad god nastupi dogadjaj gubitka, *Threshold* se postavlja na polovinu trenutne vrednosti *CongWin.*
     Posle dogadjaja istek vremena tajmera, TCP posiljalac ulazi u fazu sporog starta. Dok se nalazi u toj fazi, on eksponencijalno povecava vrednost *CongWin* sve dok *CongWin* ne dostigne vrednost *Threshold.* Kada *CongWin* dostigne *Threshold* , TCP prelazi u fazu ozbegavanja zagusenja kada *CongWin* raste linearno.
- **TCP Tahoe:** prozor zagusenja bezuslovno smanjuje na 1 MSS i prelazi u fazu sporog starta posle obe vrste dogadjaja gubitka.

- **TCP Renoe:** ne koristi fazu sporog starta nakon trostruke ACK potvrde.
- Ukidanje faze sporog starta posle trostruke ACK potvrde naziva se **brzi oporavak**.

- Ako velicina prozora iznosi *w* bajtova, a trenutno vreme povratnog puta je RTT sekundi, brzina prenosa protokola TCP je priblizno *w*/RTT.
- Ako je W vrednost *w* u trenutku kada nastupi dogadjaj gubitka, pod pretpostavkom da se RTT i W ne menjaju tokom trajanja veze, brzina TCP prenosa krece se od W/(2*RTT) do W/RTT.
- Mreza odbacuje paket iz veze kada brzina dostigne W/RTT. Brzina se tada prepolovljuje i ponovo povecava za jedan MSS/RTT za svaki RTT dok ponovo ne dostigne W/RTT.
- Prosecna propusna moc veze = (0,75*W) / RTT

- Za mehanizam kontrole zagusenja se kaze da je *fer*, ako je prosecna brzine prenosa svake veze priblizno R/K, tj ako sve veze dobijaju podjednak deo propusnog opsega linka. (R – brzina prenosa linka, K – broj TCP veza koje prolaze kroz taj link).