Transportna plast

Naloge transportne plasti

Osnova enota transportne plasti je segment.

- povezovanje dveh oddaljenih procesov
- multipleksiranje/demultipleksiranje komunikacije med procesi
- zanesljiv prenos podatkov
- kontrola pretoka in zasičenja

Storitve transportne plasti

- Logična komunikacija med aplikacijskimi procesi
  - pošiljatelj: sporočilo razbije v **SEGMENTE** in jih posreduje v enkapsulacijo omrežni plasti
  - prejemnik: dekapsulira segmente iz paketov, sestavljene segmente združi v sporočila in jih posreduje aplikacijski plasti
- Protokola TCP in UDP

Storitve so omejene s storitvami nižje plasti (npr. kanala za prenos podatkov se ne da izboljšati na transportni plasti in je ta od njega odvisna). Vsak transportni protokol lahko zagotavlja svojo množico storitev:

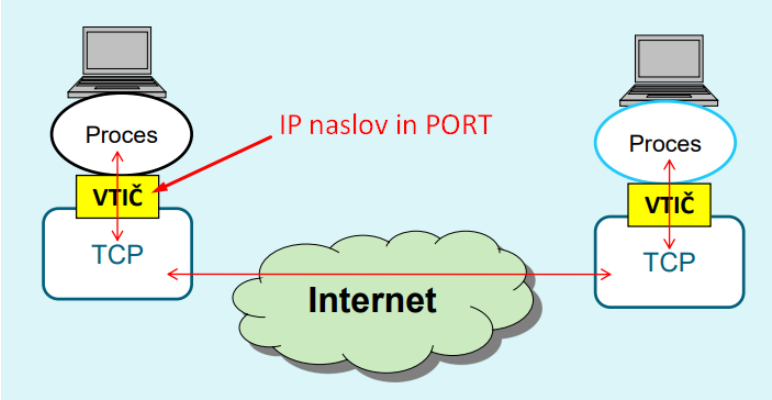
- TCP: zanesljiva, povezavna storitev z nadzorom zamašitev
- UDP: best-effortm nepovezavna storitev

Razlika med omrežno in transportno plastjo

- omrežna plast: logična povezava med končnimi sistemi
- transportna plast: logična povezava med procesi

Kako komunicirati s procesom(aplikacijo)?

- **vsak proces ima** vstopno točko, ki jo imenujemo vtič **(socket)**
- **vtič je vmesnik** med aplikacijsko in transportno plastjo
- če na končnem sistemu teče več procesov, ima vsak od njih svoj vtič
- preko vtiča proces sprejema in oddaja sporočila v omrežje



Kako poteka demultipleksiranje

Vsak transportni segment potuje znotraj svojega paketa IP. Transportni segment **ima 16-bitna podatka:** številka vrat izvora in ponora.

Kako nasloviti proces na drugi strani

za naslovitev vtiča potrebujemo:

- naslov vmesnika naprave (host address): IP številka
- naslov procesa (znotraj naprave): številka vrat

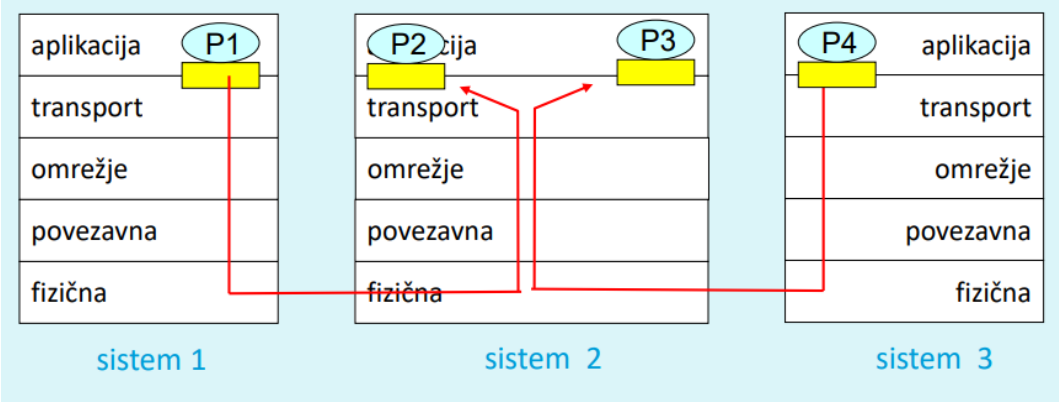
znane aplikacije uporabljajo znane številke vrat 0-1023: (t.i. well-known ports), npr.

- spletni strežnik (HTTP): 80
- poštni strežnik (SMTP): 25
- imenski strežnik (DNS): 53
- oddaljen dostop (telnet): 23
- pogovorni strežnik (IRC): 194

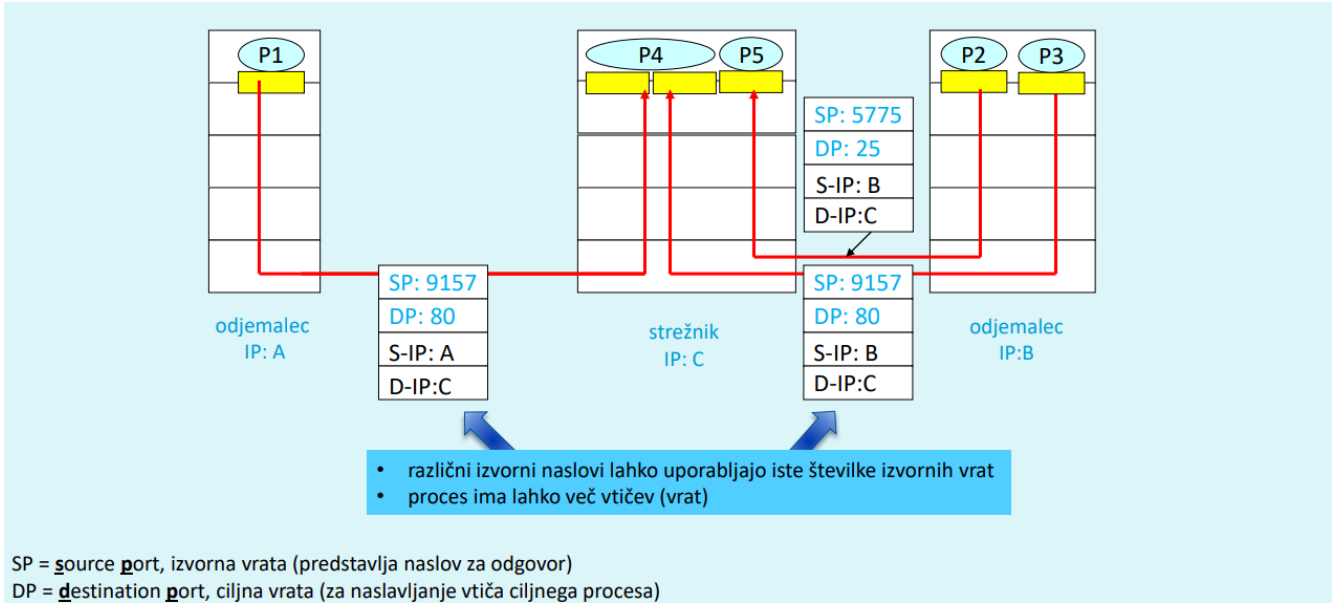
Multipleksiranje in demultipleksiranje

Popiljatelj pobira podatke z večevtičev, jih opremi z glavo in pošlje.

Prejemnik segmente razdeli ustreznim vtičem.

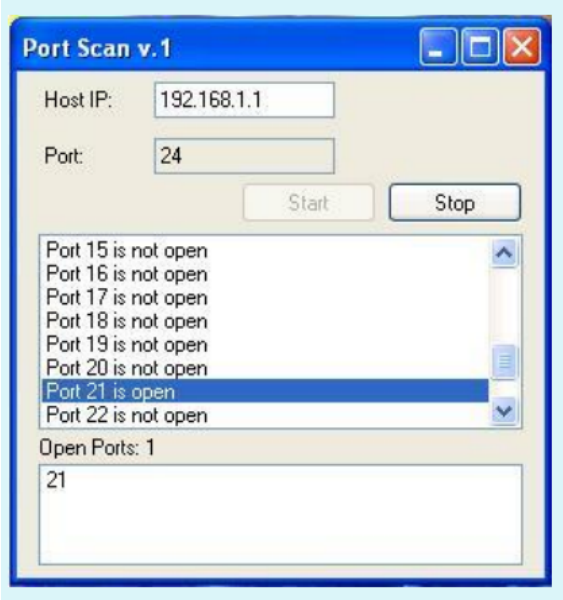


Povezavno demultipleksiranje (TCP)



Napad portscan

Napadalec izvaja *recon omrežja*. Portscan je namenjen pregledovanju strežnika, na katera vrata se bo odzival. S tem napadalec dobi vpoglede v procese, ki tečejo na strežniku. S poznavanjem šibkih točk določenih procesov (npr. SQL server) lahko ogrozi delovanje sistema.



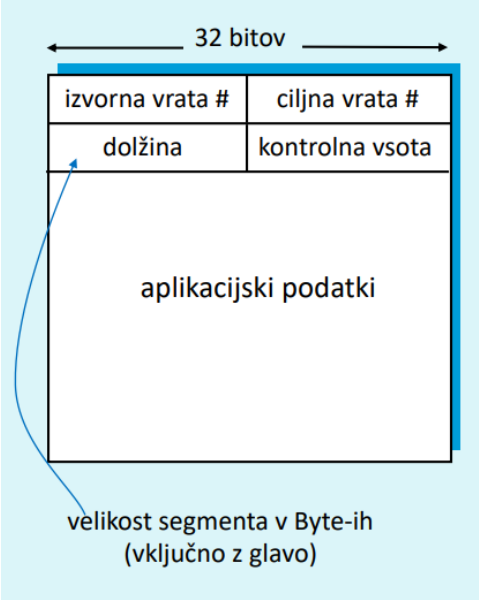
Nepovezavni transport: UDP (User Datagram Protocol)

Storitve protokola UDP

UDP datagram

namenjen uporabi v okoljih, kjer lahko toleriramo izgube in je pomembna hitrost pošiljanja:

- multimedija
- DNS, SNMP (upravljanje)
- usmerjevalni protokoli
- če pri UDP potrebujemo zanesljivost, ki je protokol ne omogoča, jo moramo zagotoviti na aplikacijski plasti



UDP: internetna kontrolna vsota

Izračun je zelo hiter. Algoritem za izračun imenujemo internetna kontrolna vsota (Internet Checksum):

- pošiljatelj sešteje 16 bitne besede in shrani eniški komplement = kontrolna vsota
- prejemnik sešteje 16 bitne besede skupaj s kontrolno vsoto -> dobiti mora same enice

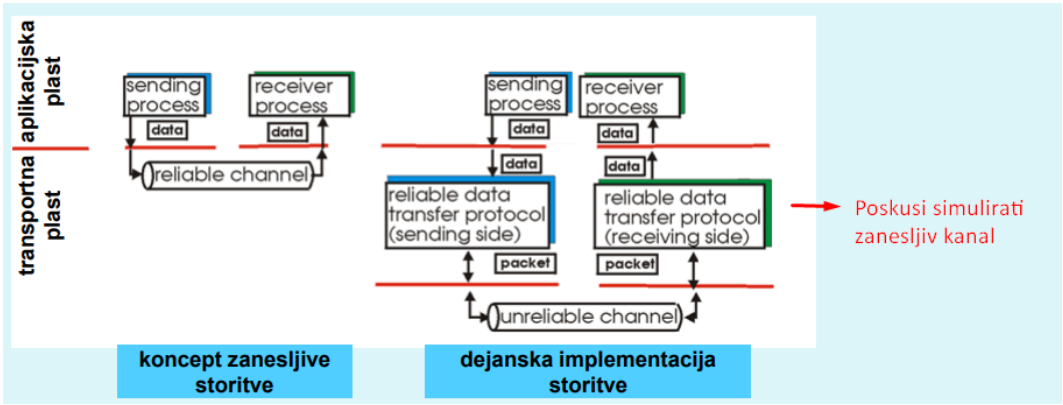
Problem: seštevanje je kumulativno. Napadalec lahko premeša 16 bitne besede(zamenja vrstni red bitov) in bo kontrolna vsota ok čeprav so podatki spremenjeni.

Zakaj datagram vsebuje kontrolno vsoto?

- Protokol ne zagotavlja zaznavanje in odpravljanje napak.
- Do napak lahko pride tudi pri hranjenju segmenta v spominu usmerjevalnika, protokol pa zagotavlja le zaznavanje napak pri prenosu).
- UDP kontrolna vsota je namenjena preverjanju pravilnosti med izvornim in ciljnim procesom, ne pa pri potovanju po povezavah (t.i. princip končnih sistemov, end-to-end argument/principle).

Osnutek protokola TCP (osnovno ogrodje)

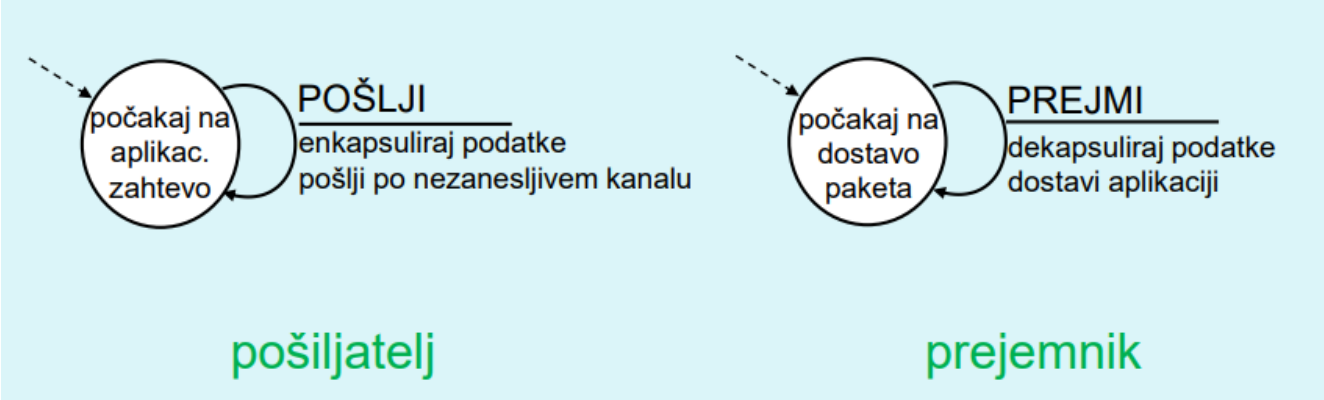
Potrebujemo protokol, ki zagotavlja zanesljivo dostavo(tu poskrbimo za kar ni poskrbel IP) z uporabo nezanesljivega kanala. Podatki se ne smejo okvariti, izgubiti ali dostaviti v napačnem zaporedju.



Konstruirajmo protokol, ki bo omogočal:

- **1. osnovno funkcionalnost**: pošiljanje in prejemanje podatkov
- **2. reševanje napak** pri prenosu (potrjevanje z ACK in NAK)
  - poenostavljeno potrjevanje (samo pozitivne potrditve ACK)
  - optimizirano potrjevanje (skupinsko potrjevanje paketov!)
- **3. obravnavo izgubljenih** paketov (ponovno pošiljanje)
  - odpornost na izgubljene potrditve paketov

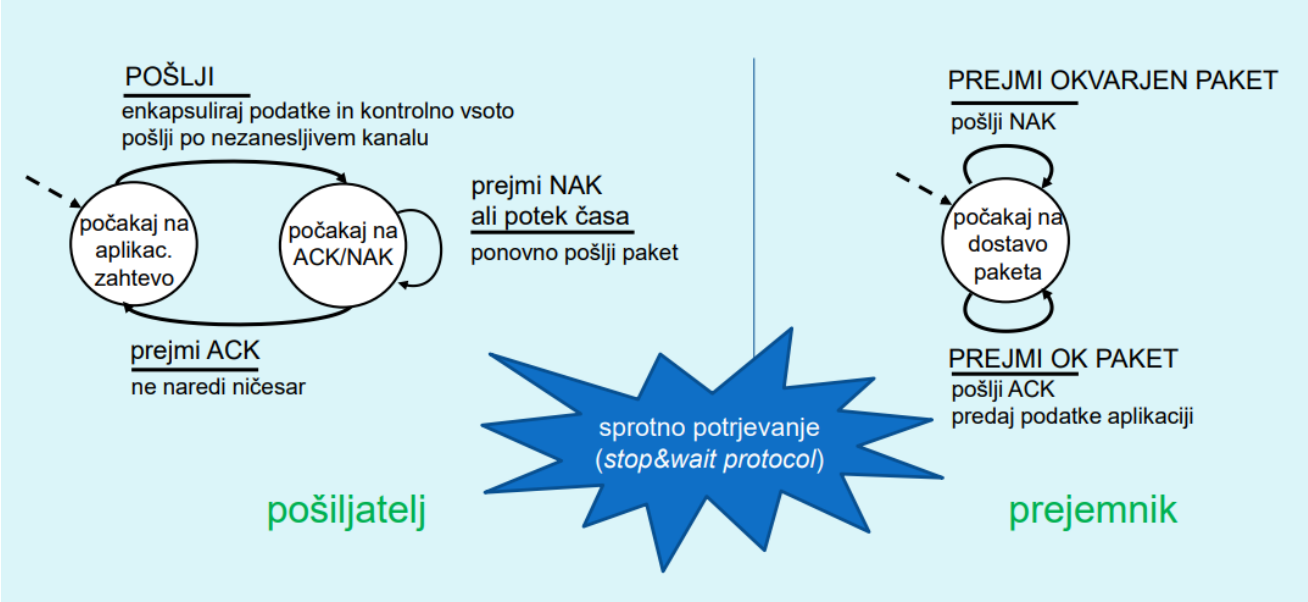
1. Enostavni protokol



2. Reševanje iz napak pri prenosu

Vsebina paketov se lahko okvari, kar zaznamo s kontrolno vsoto.

- Dodamo funkcionalnosti:
  - zaznavanje napak
  - potrditve, ali je bil paket sprejet pravilno (ACK) ali nepravilno (NAK)
  - čakanje na potrditev(časovni interval)
  - ponovno pošiljanje ob napaki

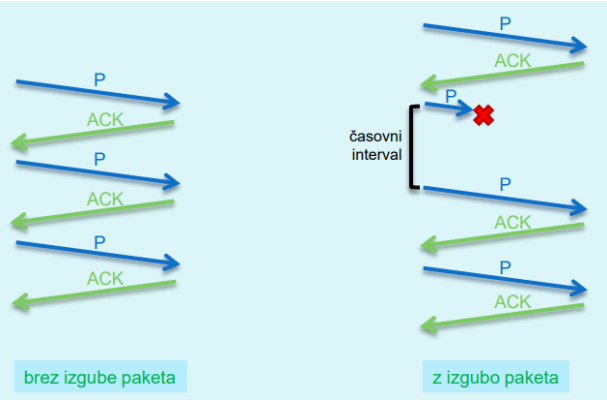


Pošiljatelj lahko le ćaka dokler ne dobi ACK in po potrebi ponovno pošlje, drugo ne more. Sprotno potrjevanje je način preverjanja, TCP ne deluje čisto tako, saj lahko pošlje več pakeetov in čaka na potrjevanje.

3. Izguba paketov ali potrditev

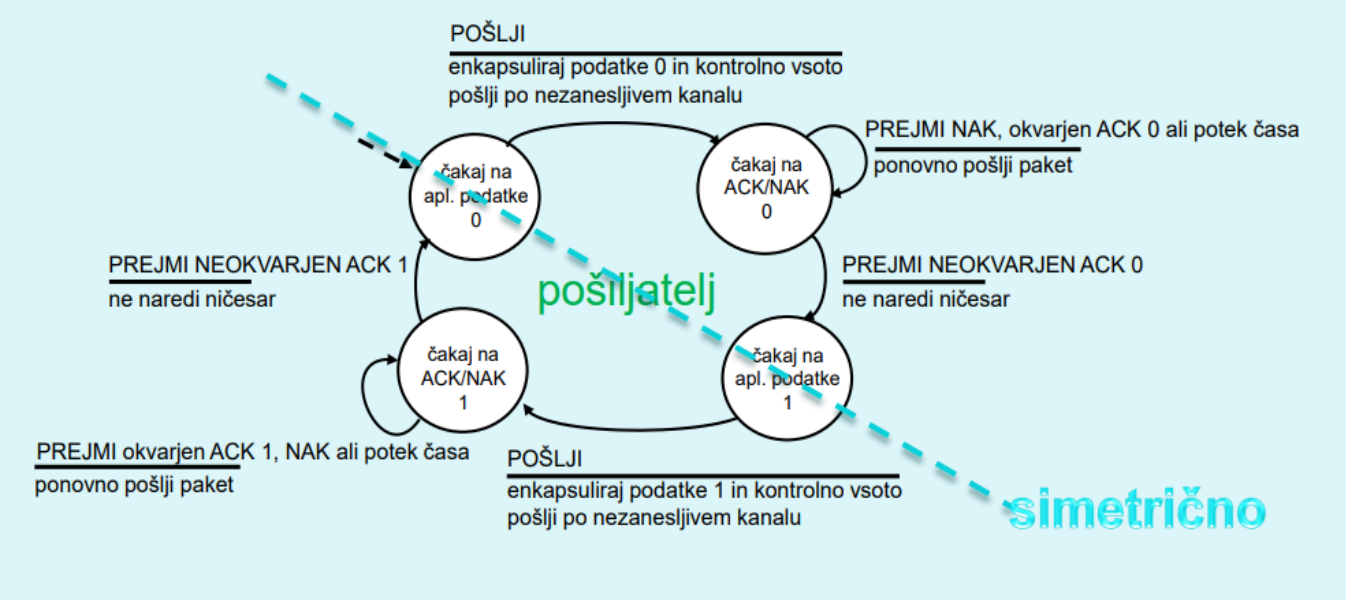
Novi problem: paketi se lahko izgubijo in ne pridejo do prejemnika, kar zaznamo s tem, da jih ta ne potrdi.

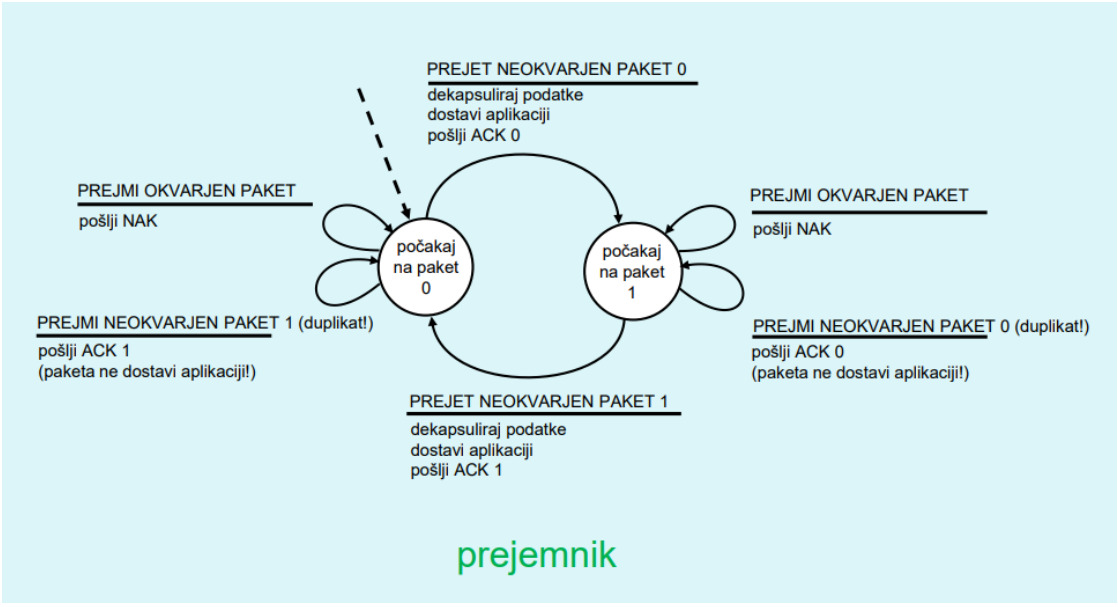
Rešitev: pošiljatelj počaka določen časovni interval(potrebujemo štoparico) na ACK in če ga ne prejme, pošlje paket ponovno.



Možna je tudi **izguba potrditve** ali **prekratek časovni interval**. V obeh primerih pride do prejetja podvojenega paketa. To rešimo s **številčenjem paketov** in zaznavanjem istih številk.

Za številčenje paketov sta dovolj 2 števili(npr. sodo/liho), saj moramo vedeti le če je paket enak prejšnjemu.





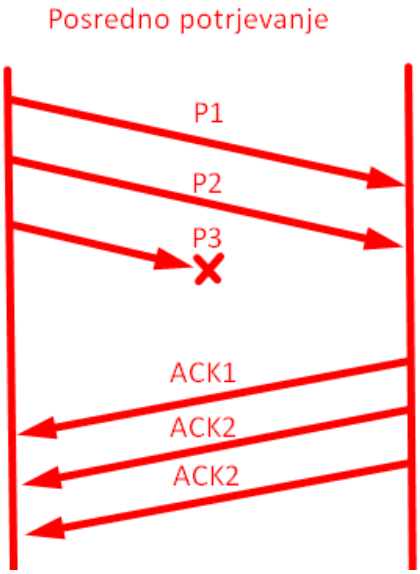
4. Izboljšava: posredno potrjevanje(samo ACK)

Enak učinek potrjevanja dosežemo, če uporabimo samo ACK, v katerega vključimo številko segmenta, ki ga potrjujemo,

- pri vsakem prejetem segmentu (pravilno sprejetem ali okvarjenem), prejemnik odgovori z ACK za zadnji še uspešno prejeti segment,
- če pošiljatelj prejme ACK za isti segment dvakrat, to pomeni, da segment, ki je sledil, ni bil sprejet pravilno (torej enako kot NAK)

Neposredno potrjevanje: uporaba ACK in NAK

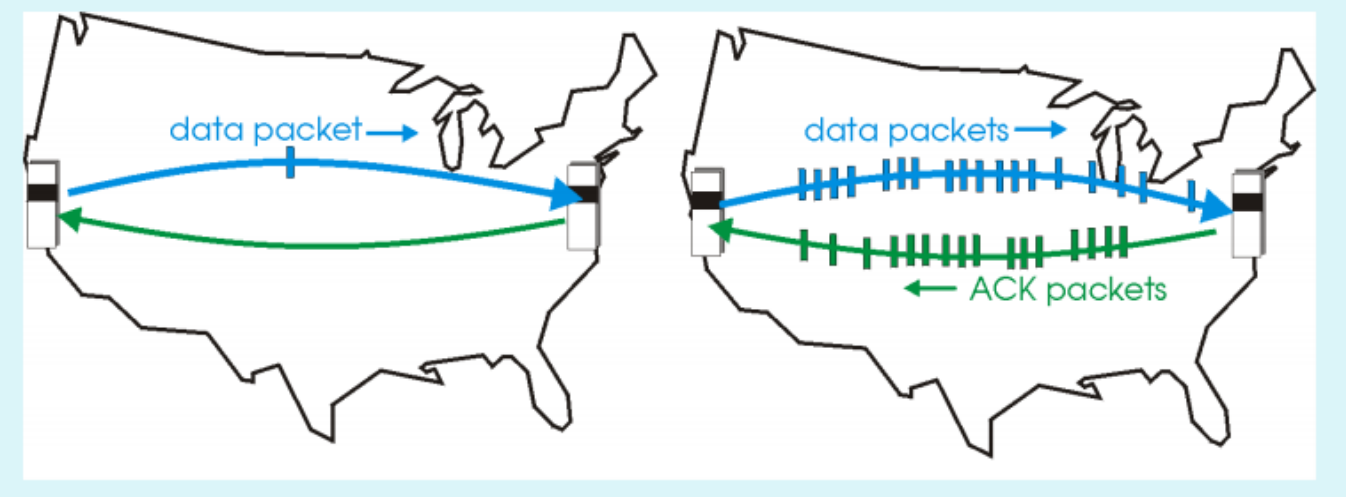
Posredno potrjevanje: uporaba samo ACK



5. Reševanje neučinkovitosti sprotnega potrjevanja

Problem: pošiljatelj, ki uporablja sprotno potrjevanje (stop&wait protocol), mora pred oddajo naslednjega paketa čakati na potrditev prejšnjega.

Ideja: namesto zaporednega pošiljanja, implementirajmo vzporedno oziroma tekoče (cevovodno, pipelined) pošiljanje, ker lahko istočasno potuje več paketov, brez sprotnega čakanja na njihovo potrditev, je izkoriščenost kanala večja.



5. Tekoče pošiljanje

Potrebujemo:

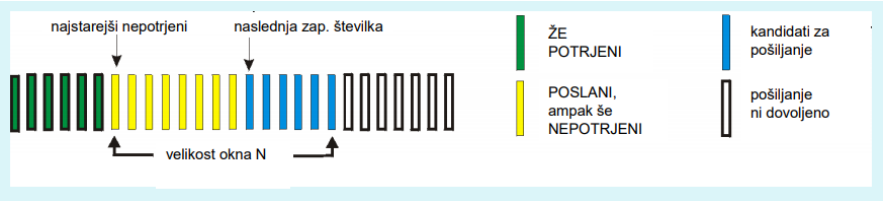
- večji razpon števil za številčenje paketov (0 in 1 ni več dovol za ločevanje paketov, saj ni sprotnega potrjevanja)
- shranjevanje paketov(pomnilnik) na strani pošiljatelja in prejemnika

Poznamo dve obliki protokolov za tekoče pošiljanje:

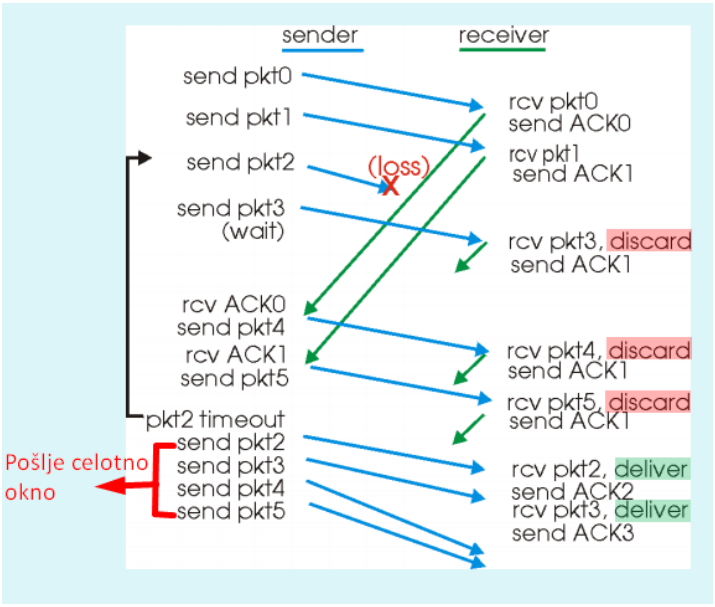
- ponavljanje **N nepotrjenih (go-back-N)**
- ponavljanje **izbranih (selective repeat)**

Ponavljanje n nepotrjenih

- Pošiljatelj hrani "okno" največ dovoljenih nepotrjenih paketov. Tak protokol imenujemo tudi protokol z drsečim oknom
- Ko prejemnik pošlje ACK(n), potrdi s tem vse pakete do vključno n
- Časovna kontrola za najstarejši paket
- Ko časovna kontrola poteče, ponovno pošlji vse nepotrjene pakete v oknu

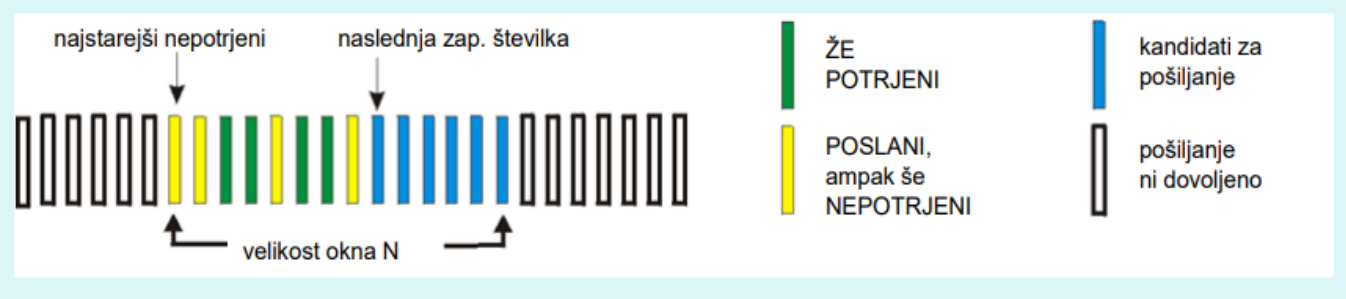


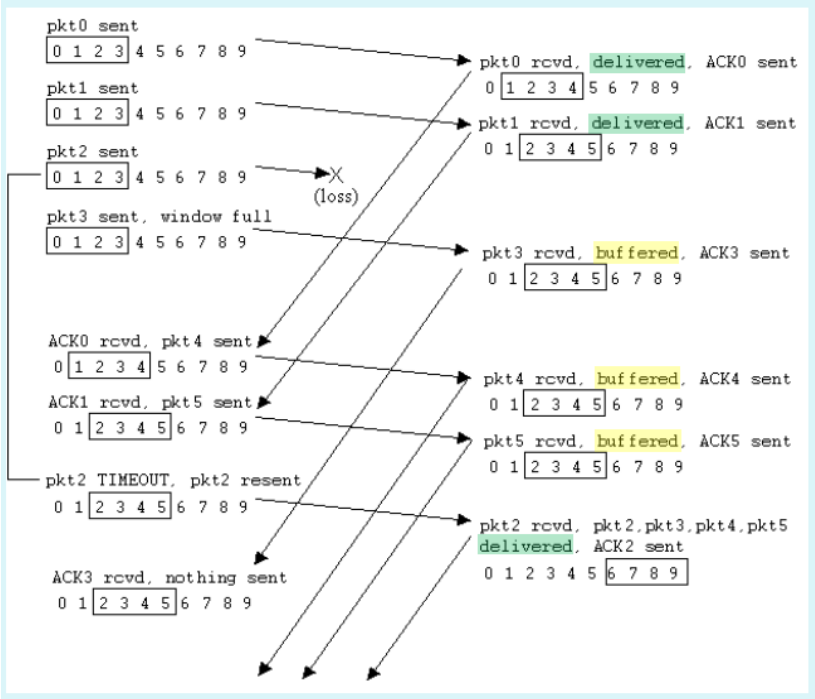
- Če se 1 paket zgubi, neprestano pošilja ACK od zadnjega uspešno prejetega paketa. To pri pošiljatelju povzroči timeout in ta spet pošlje celotno okno.
- Prejemnik lahko prejme podvojene pakete, a jih zazna po zaporednih številkah in zavrže.
- Prejemnik lahko prejme pakete v napačnem vrstnem redu. Te zavrže, saj bodo poslani ponovno, ko poteče štoparica za najstarejšega.
- Če se ACK zgubi ni pomembno. Prejemnik shrani pakete, naslednji ACK pa vedno potrdi vse za nazaj do vključno trenutno prejetega paketa.
- Ker se paketi ob napaki zavržejo, prejemnik ne potrebuje pomnilnika za vrstni red, saj jih bo dobil ponovno, ko poteče štoparica.



Ponavljanje izbranih

- Prejemnik potrjuje vsak prejeti paket posamezno
- Prejemnik shranjuje pakete, prejete v napačnem vrstnem redu, in jih sortira pred dostavo aplikaciji
- Pošljatelj **ponovno pošlje samo tiste pakete, za katere ni dobil ACK**
- Pošiljatelj hrani **štoparico za vsak posamezni nepotrjeni paket**
- Ni več kumulativnega potrjevanja - vsak ACK potrdi le svoj paket





Potrjevanje

Možne so vse kombinacije potrjevanja

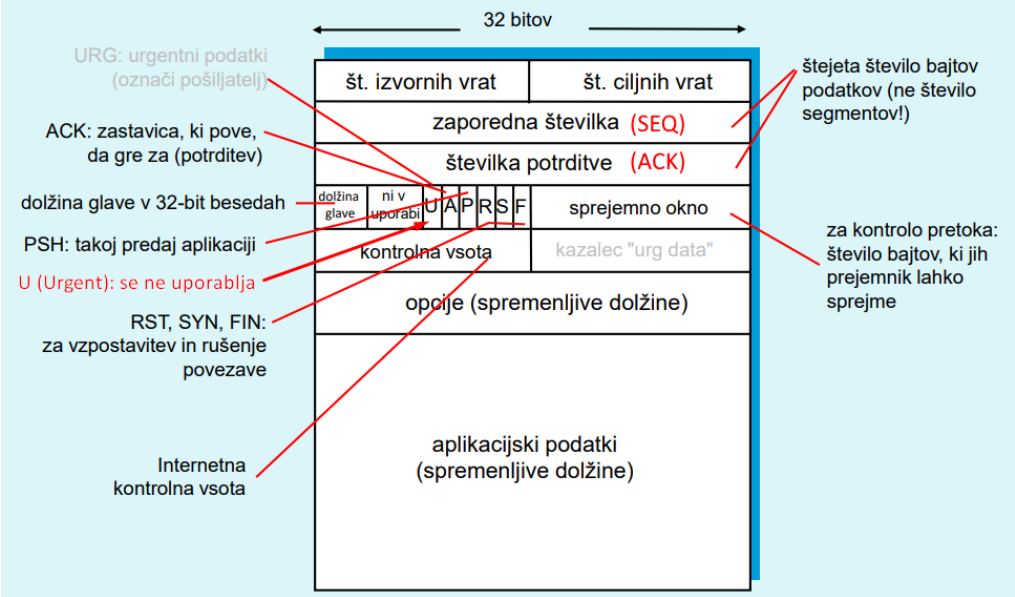
- neposredno: ACK in NAK, posredno: samo ACK
- sprotno: za vsak paket sproti, tekoče: z uporabo drsečega okna za več paketov

Protokol TCP

Lastnosti protokola TCP

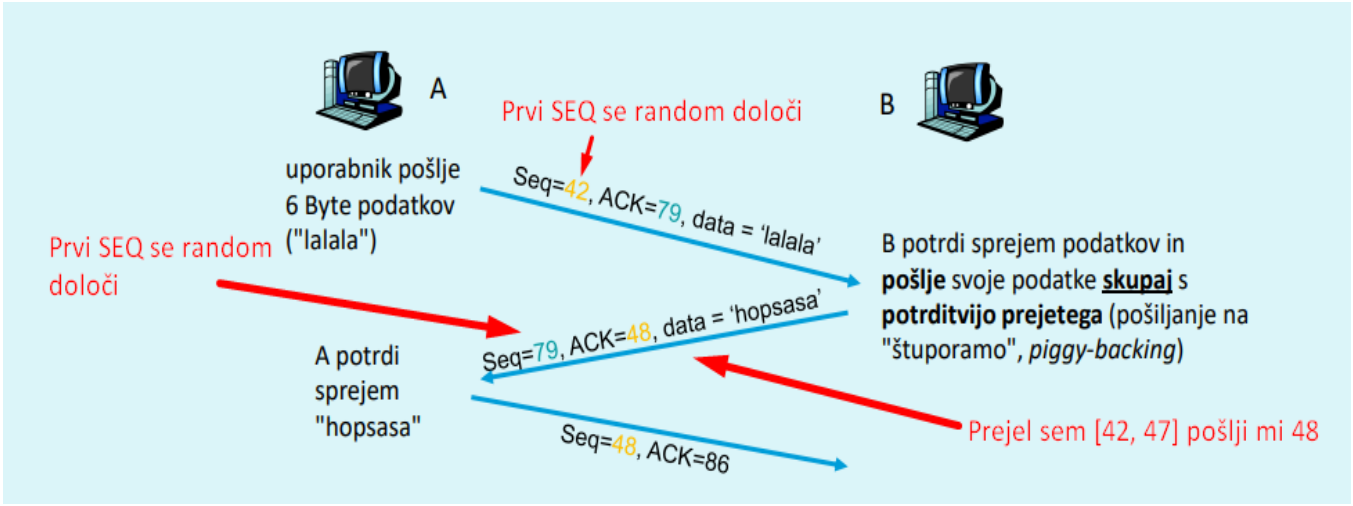
- Izvaja se med dvema točkama (point-to-point): **en pošiljatelj, en sprejemnik** • Je **povezavni protokol** (uporablja vzpostavitev/rušenje zveze) • Izvaja **dvosmerni promet** pri povezavi (full duplex, MSS) • Nudi **zanesljiv**, urejen tok podatkov • ima **kontrolo pretoka** (angl. flow control, pošiljatelj ne preobremeni prejemnika) • ima **kontrolo zasičenja** (angl. congestion control, pošiljatelj ne preobremeni omrežja) • uporablja **tekoče pošiljanje**, velikost okna se avtomatsko določa glede na kontrolo pretoka in kontrolo zasičenja

TCP segment



TCP: številčenje segmentov in potrditev

- Pošiljatelj in prejemnik najprej **vzpostavita zvezo**. Povezava je nato dvosmerna (vsak lahko pošilja drugemu).
- Pošiljatelj lahko v enem segmentu istočasno pošlje nove podatke in potrditev (ACK) prejšnjega segmenta
- **SEQ (zaporedna številka):** številka prvega Byte-a v segmentu
- **ACK (potrditev):** številka naslednjega pričakovanega Byte-a

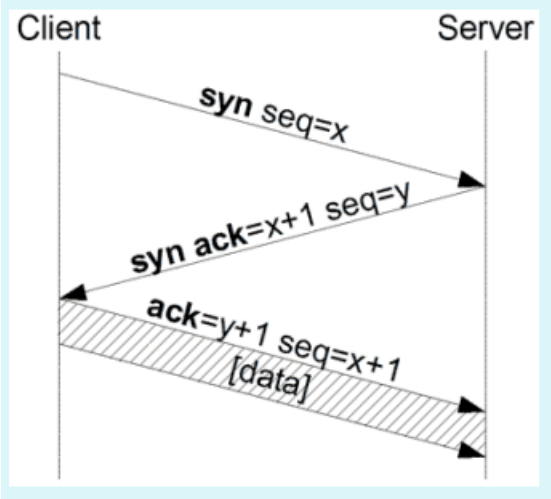


TCP: vzpostavljanje povezave

Pošiljatelj in prejemnik pred pošiljanjem izvedeta rokovanje (handshake), v katerem izmenjata parametre: • **začetne pričakovane zaporedne številke** (naključno določene) • **velikosti medpomnilnikov** (za kontrolo pretoka)

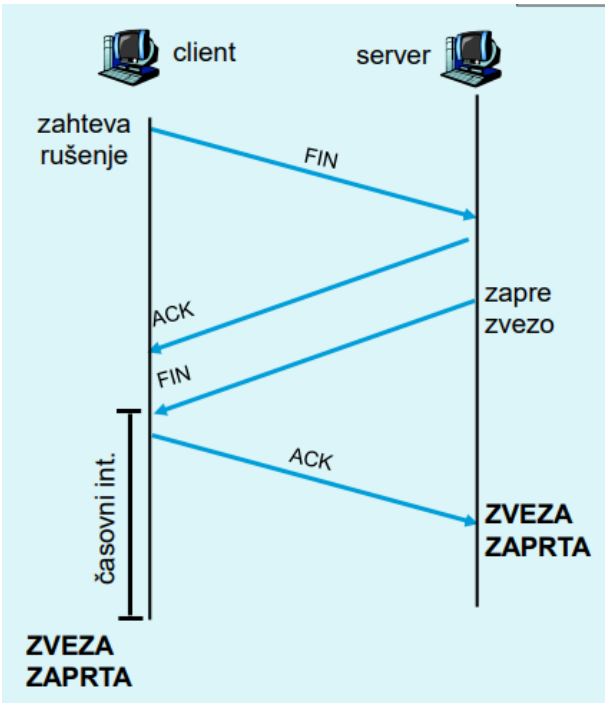
Trojno rokovanje (three-way handshake):

1. Odjemalec pošlje segment z zastavico **SYN** (sporoči začetno številko segmenta, ni podatkov)
2. Strežnik vrne segment **SYN ACK** (rezervira medpomnilnik, odgovori z začetno številko svojega segmenta)
3. Odjemalec vrne **ACK**, lahko že s podatki (piggy-backing)

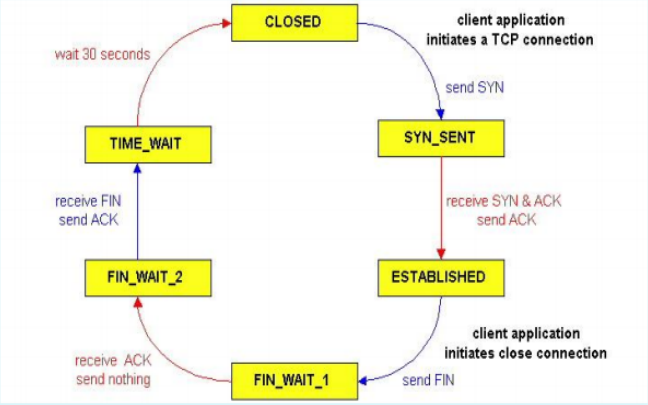


TCP: rušenje povezave

1. odjemalec pošlje segment TCP FIN strežniku
2. strežnik potrdi z ACK, zapre povezavo, pošlje FIN
3. odjemalec prejme strežnikov FIN, potrdi ga z ACK. Počaka časovni interval, da po potrebi ponovno pošlje ACK, če se ta izgubi
4. strežnik sprejme ACK. Zveza se zapre šele, ko čas poteče. Če ne bi čakal, bi lahko server poslal FIN, ki bi lahko priletel v neko drugo, novo povezavo.

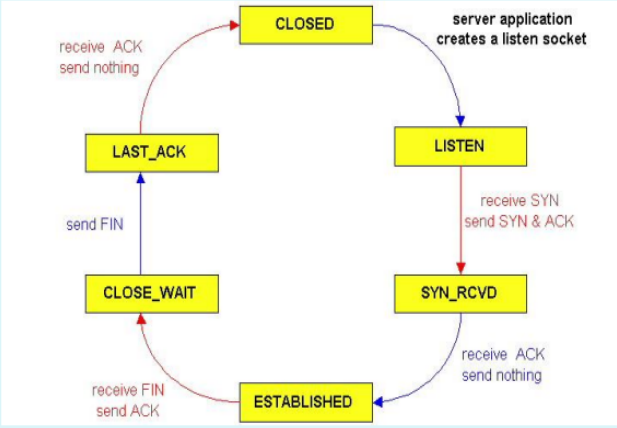


Življenjska cikla odjemalca in strežnika



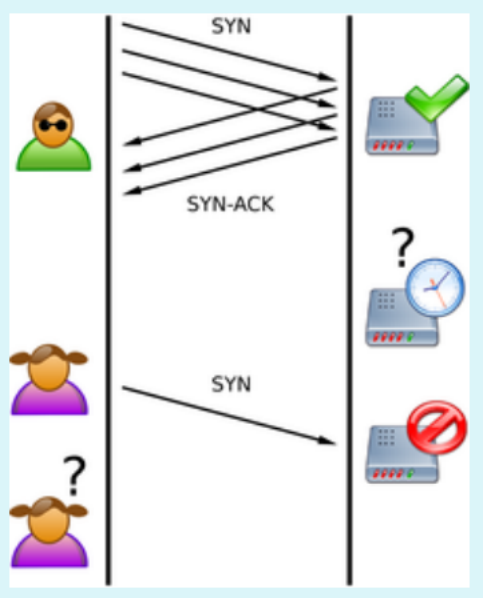
TCP odjemalec

TCP strežnik



Napad SYN FLOOD

- Danes večinoma saniran
- Napad, v katerem napadalec pošlje strežniku veliko število paketov za vzpostavitev zveze (TCP SYN), pri čemer strežnik vsakič rezervira del svojega medpomnilnika
- Pomnilnik ostane zaseden zaradi napol odprtih zvez (napadalec ne zaključi tretjega koraka rokovanja z ACK). Zaradi velikega števila odprtih povezav strežniku zmanjka prostora in pride do odpovedi sistema (angl. denial of service)
- Možno izvesti tudi porazdeljeno - pošiljanje TCP SYN iz več virov
- Rešimo z uvedbo timeout-a ali z IDS(Intrusion Detection System) napravami, ki zaznavajo vdore



Nastavitev časovne kontrole

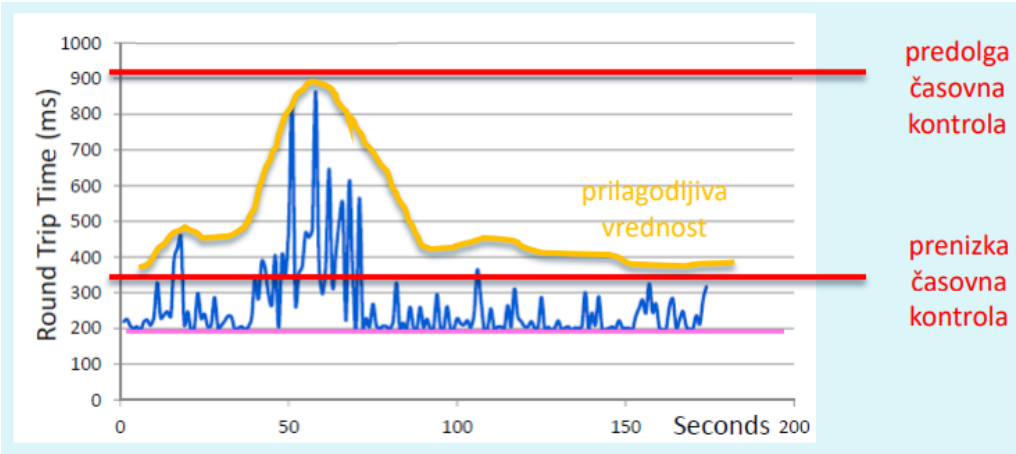
Časovna kontrola (štoparica): potrebna za uporabo zanesljive dostave, t.j. ponovnega pošiljanja, če se izgubi paket ali njegova potrditev

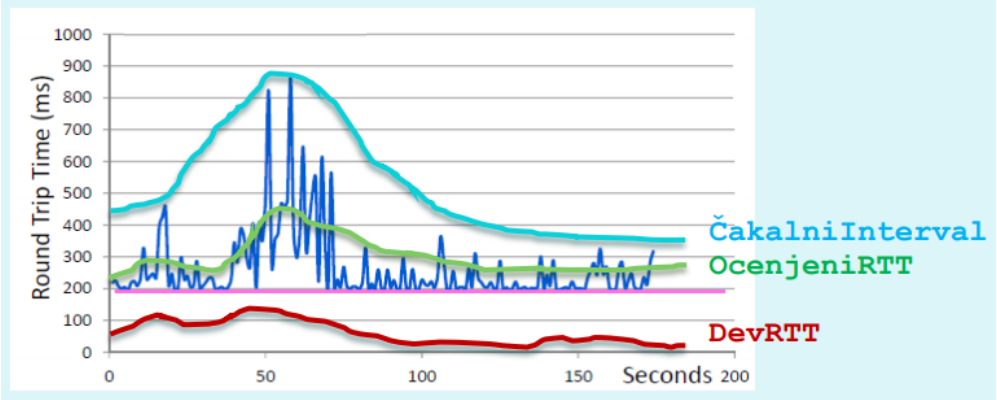
Kako nastaviti dolžino čakalnega intervala?

- interval mora biti daljši od časa vrnitve (RTT, Round Trip Time) = čas za pot paketa od pošiljatelja do prejemnika in nazaj
- če je prekratek, imamo preveč ponovnih pošiljanj
- če je predolg, prepočasi reagiramo na izgubljene segmente

Primer ocenjevanja RTT

- Avtomatsko opravimo meritve RTT (round-trip time) od pošiljanja segmenta do prejema potrditve, da ocenimo smiselno velikost časovne kontrole
- Izmerjen RTT je lahko nestabilen zaradi različnih poti in obremenjenosti usmerjevalnikov!
- Potrebujemo "prilagodljivo vrednost" časovne kontrole
- Računamo rekurzivno, da ni potrebno hraniti vseh prejšnjih meritev





- izračunamo gibajoče povprečje
  `OcenjeniRTT[i] <- (1-α)*OcenjeniRTT[i-1] + α*IzmerjeniRTT[i]`
  običajno uporabimo: α=0.125
- izračunamo gibajočo deviacijo
  `DevRTT[i] = (1-β)*DevRTT[i-1] + β*|IzmerjeniRTT[i] - OcenjeniRTT[i]|`
  običajno uporabimo β=0.25
- vrednost čakalnega intervala TCP nastavi kot OcenjeniRTT + "rezerva":
  `ČakalniInterval[i] = OcenjeniRTT[i] + 4*DevRTT[i]`

Način potrjevanja TCP

TCP uporablja **kombinirano rešitev** ponavljanja N nepotrjenih in potrjevanja posameznih. Imamo **štoparico za najstarejši nepotrjeni segment**, vendar se **ob poteku časovne kontrole** ne **pošlje** celega okna, temveč le **najstarejši nepotrjeni segment**.

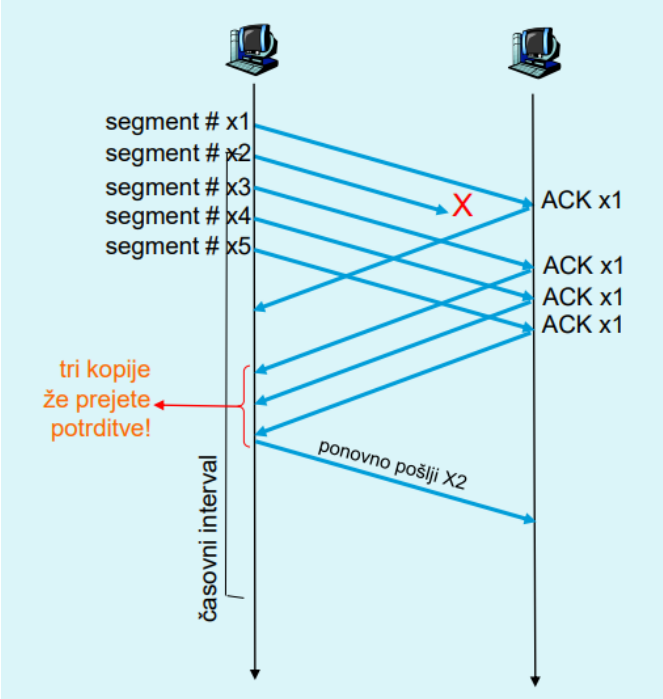
Posebnosti pri potrjevanju TCP

Dogodek pri prejemniku	Odziv prejemnika
Sprejem segmenta s pričakovano številko, vsi prejšnji že potrjeni.	Počakaj na naslednji segment max 500 ms. Če ta pride v tem intervalu, izvedi **zakasnjeno potrditev obeh** (delayed ACK). Če ne pride v tem intervalu, potrdi samo prejetega.
Isto kot zgoraj, a potrditev za prejšnji segment še ni bila poslana.	Takoj pošlji **kumulativno potrditev** za oba segmenta brez izvajanja zakasnjene potrditve.
Sprejem segmenta s previsoko številko (**zaznamo vrzel**)	Takoj potrdi zadnji še sprejeti segment (pošlji duplikat ACK).
Sprejem segmenta z najnižjo številko iz vrzeli (**polnjenje vrzeli**)	Takoj potrdi segment.

Hitro ponovno pošiljanje (fast retransmit)

Ponovno pošiljanje se običajno izvede po preteku časovne kontrole, včasih je časovni interval predolg in ga lahko v določenih situacijah skrajšamo.

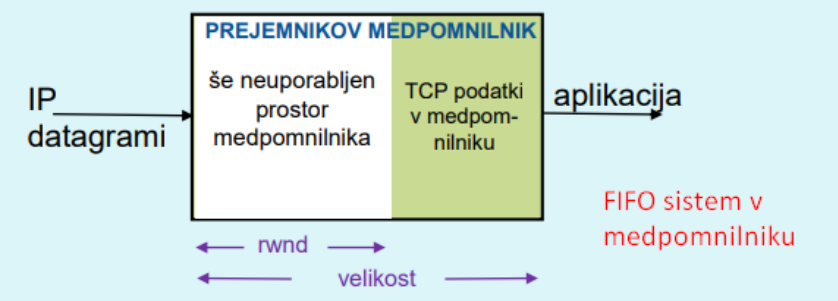
Hitro ponovno pošiljanje pošiljatelj izvede **pred potekom časovnega intervala, če prejme za nek paket 3 podvojene potrditve.**



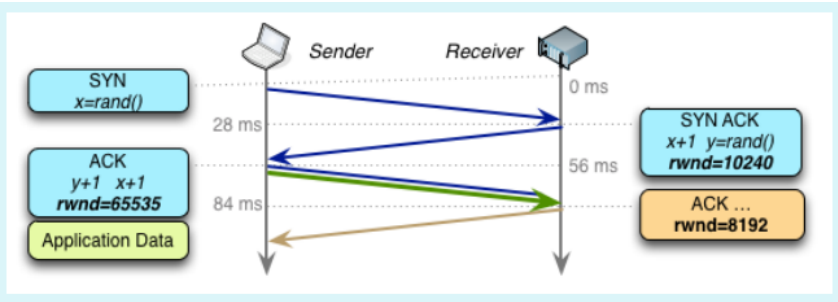
TCP: Kontrola pretoka

Uporablja se za usklajevanje hitrosti med pošiljateljem in prejemnikom: pošiljatelj ne sme pošiljati hitreje, kot lahko prejemnik bere, da ne povzroči prekoračitve medpomnilnika (prejemnikov prostor, kjer se začasno shranjujejo prejeti segmenti pred predajo aplikaciji).

- neuporabljen (razpoložljiv) prostor medpomnilnika: rwnd = velikost - [LastByteRcvd - LastByteRead]



- prejemnik sporoča pošiljatelju velikost razpoložljivega prostora v glavi vsakega segmenta (rwnd) - receiver window
- pošiljatelj ustrezno omeji število paketov, za katere še ni prejel potrditve



Nadzor zasičenja (nadzor zamašitve/congestion control)

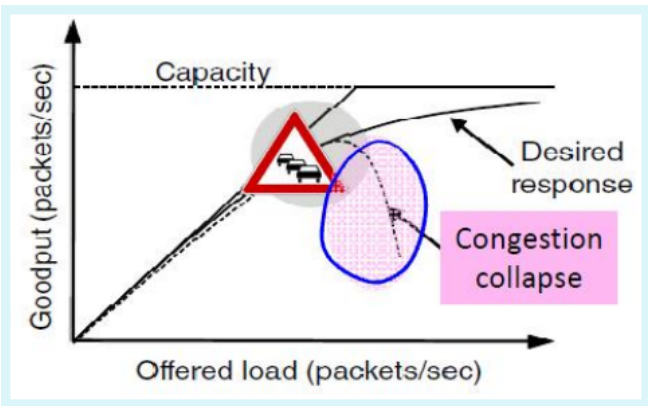
Zasičenje: stanje omrežja, ko veliko virov naenkrat prehitro pošilja preveč podatkov za dano omrežje

Posledice zasičenja:

- izguba segmentov (prekoračitve medpomnilnika v usmerjevalnikih)
- velike zakasnitve (čakalne vrste v usmerjevalnikih)

Ni isto kot nadzor pretoka:

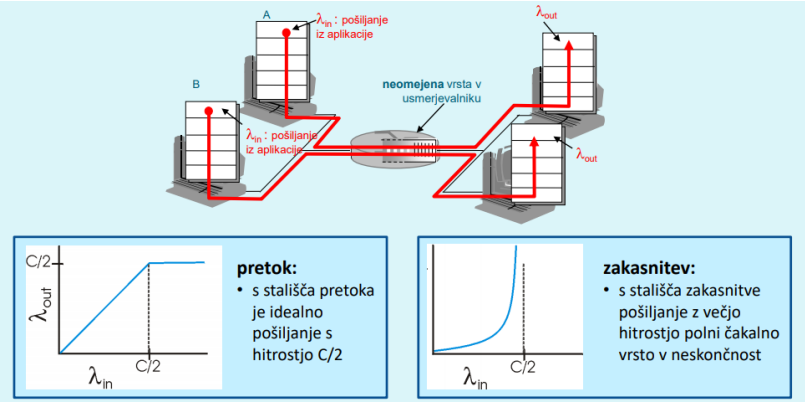
- kontrola pretoka: da ne zasujemo prejemnika
- kontrola zasičenja: da ne zasujemo omrežja



Primeri

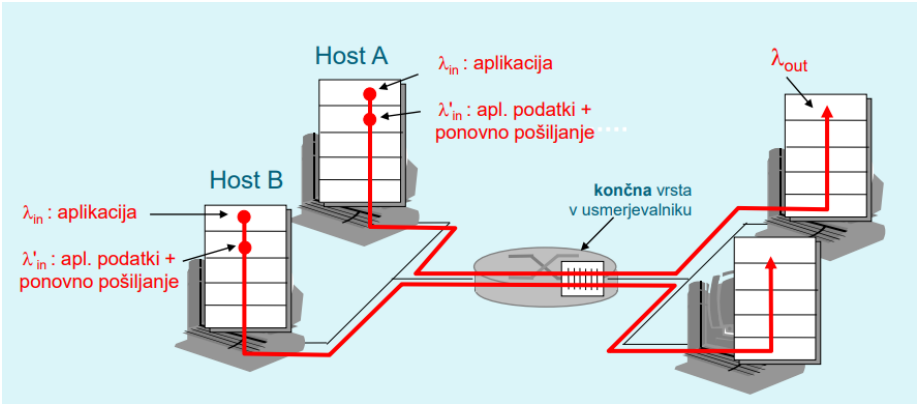
Fiktiven primer

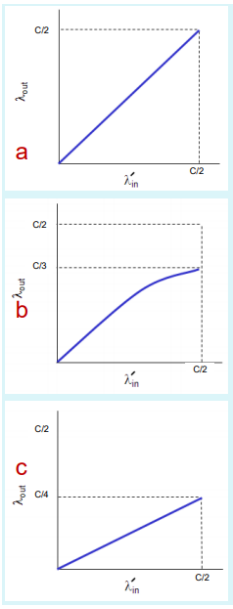
- dva pošiljatelja, neomejen pomnilnik v usmerjevalniku(za čakalno vrsto), C = kapaciteta kanala



Preprost primer

- končna vrsta
- ponovna pošiljanja segmentov zaradi izgub in zakasnitve





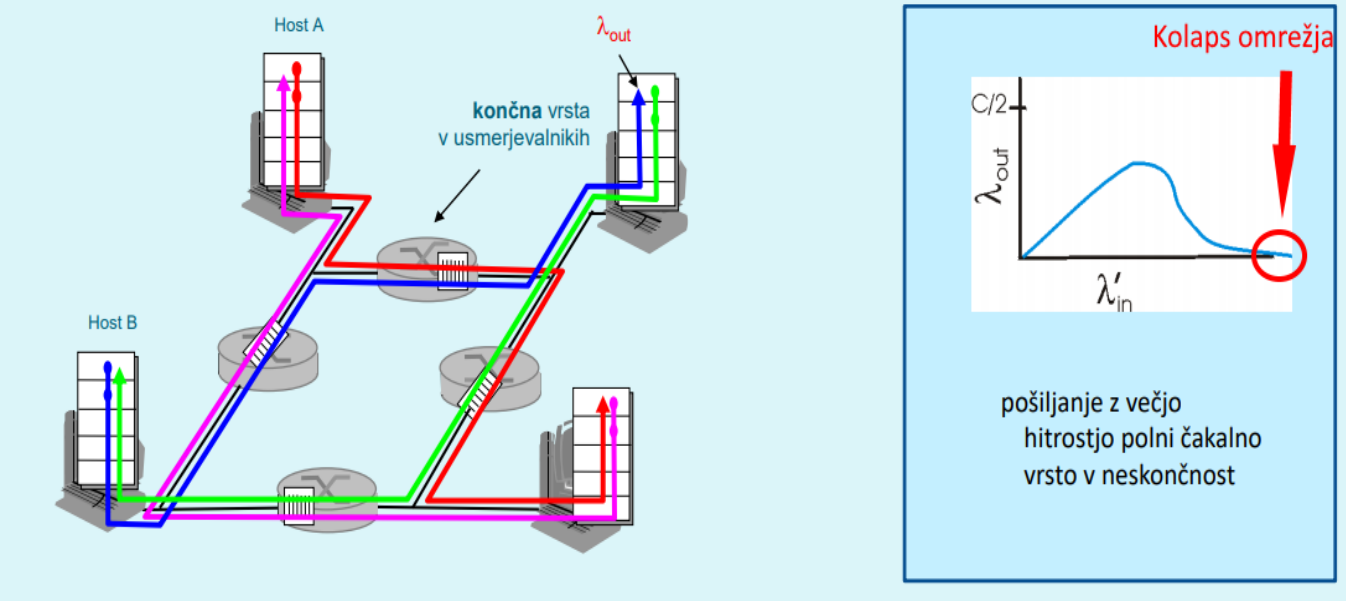
a) Segment oddamo le, ko je prostor v vrsti, tako da ni izgub. To v praksi ni možno, ker pošiljatelj ne ve stanja na usmerjevalniku.

b) dogajajo se izgube paketov in posledično ponovna pošiljanja.

c) ponovna pošiljanja zaradi izgube paketov in velikih zakasnitev.

Realen primer

- daljše poti, če se paket izgubi na n-tem skoku, so bili vsi dotedanji prenosi zamanj.

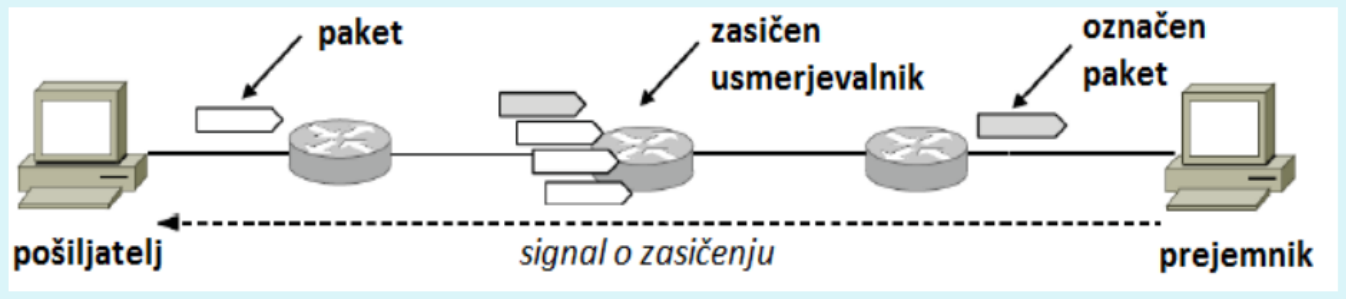


Na grafu je označen kolaps omrežja. Če pride do te situacije, omrežje obstane in je edina rešitev, da se celotno omrežje ugasne in znova prižge. Ta primer je bil prvič zabeležen na predhodniku interneta - ARPANET-u.

Kako nadzorovati zasičenja?

Dva pristopa:

- z uporabo **omrežnih storitev:** usmerjevalniki v omrežju obvestijo pošiljatelja, da je prišlo do zasičenja
  - uporaba direktnega obvestila o zasičenju (ECN – explicit congestion notification)
  - usmerjevalnik nastavi ustrezen bit in sporoči sprejemljivo hitrost oddajanja (npr. pri ATM - ta ne deluje po best-effort modelu)
- na podlagi **končnih sistemov** (end-to-end):
  - bodisi prejemnik sporoči pošiljatelju, da so usmerjevalniki na poti sporočili zasičenje
  - bodisi pošiljatelj opazuje čas do prejema potrditve (to tehniko uporablja TCP)

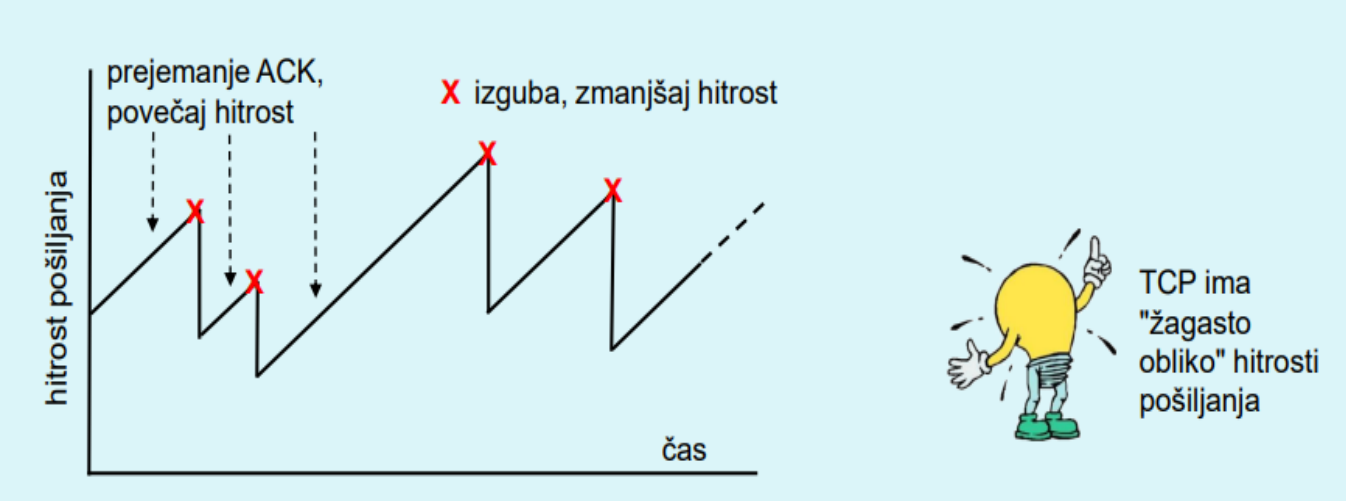


TCP: Nadzor zasičenja

- **Ideja:** pošiljatelj želi **pošiljati čim hitreje**, vendar še **pod mejo zasičenja omrežja.**

Kako najti pravo hitrost?

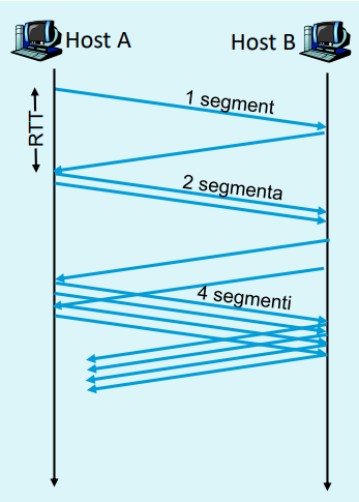
- **Rešitev:** vsak pošiljatelj si sproti nastavlja hitrost na podlagi opazovanja reakcij v omrežju na pošiljanje:
  - če prejme potrditev (ACK), ni zasičenja, poveča hitrost
  - če se segment izgubi, je to posledica zasičenja, zmanjšaj hitrost



- Okno rwnd (receive window) smo že spoznali (omejitev količine nepotrjenih podatkov za kontrolo pretoka)
- za nadzor zasičenja uporabljamo okno cwnd (congestion window).
- možni dogodki:
  - **POZITIVEN:** prejem ACK: povečuj cwnd
    - eksponentno (x2, počasni začetek, slow start) ali
    - linearno (+1, izogibanje zasičenju, congestion avoidance)
  - **NEGATIVEN:** potek časovnega intervala (segment se izgubi):
    - zmanjšaj cwnd na 1
- **cwnd (Congestion Window)** je nastavitev pri pošiljatelju in se ne pošilja. Je omejitev, ki pove **koliko podatkov lahko pošljemo, da ne zasujemo omrežja**.
- **rwnd (Receiver Window)** je omejitev, ki pove **kako hitro lahko pošiljamo, da ne zasujemo prejemnika**
- **TCP torej pošilja s hitrostjo, ki ustreza min(rwnd, cwnd)**

1. faza: počasen začetek (Slow Start)

1. ob vzpostavitvi povezave: velja cwnd = 1 segment
2. hitrost povečuj eksponentno:
    - za vsak prejeti ACK: cwnd <- cwnd + 1
    - (oziroma po vseh prejetih ACK izgleda kot eksponentna rast cwnd <- cwnd * 2)
3. ko pride do prve izgube, se ustavi in si zapomni PRAG (polovica trenutnega cwnd, ko pride do zasičenja) ter nastavi cwnd=1
4. izvajaj počasen začetek od koraka 1. Ko prideš do vrednosti PRAG, preidi v način izogibanja zasičenju.

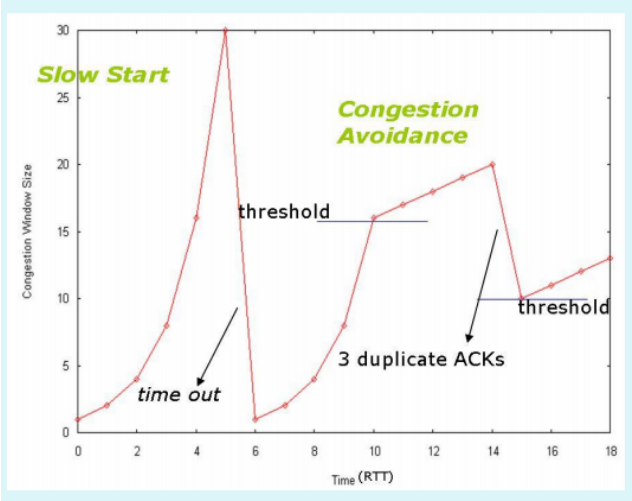


2. faza: izogibanje zasičenju (Congestion Avoidance)

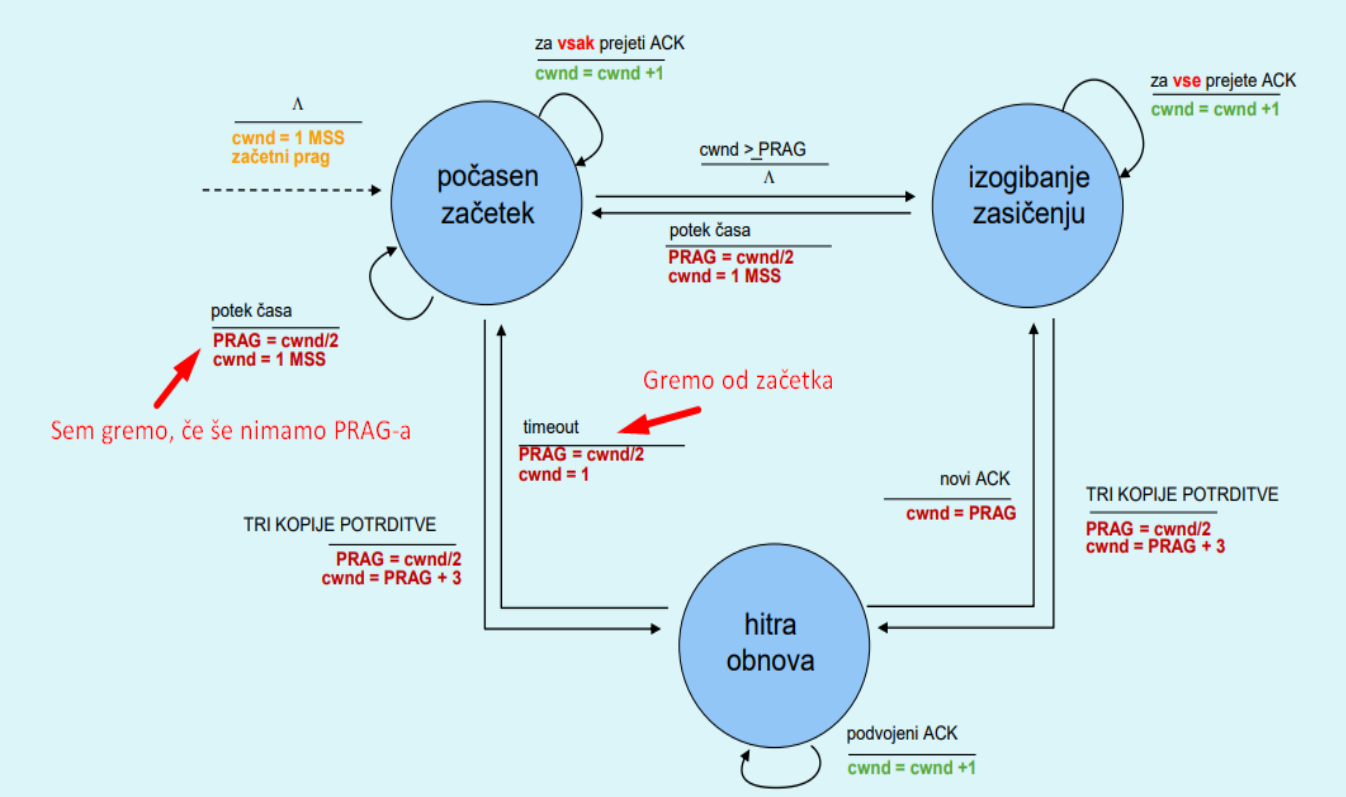
Kadar cwnd > PRAG, povečuj cwnd linearno za 1 MSS. Na ta način se bolj počasi približaj pragu zasičenja.

MSS(Maximum Segment Size) je največja velikost podatkov v TCP segmentu. MTU je največja velikost podatkov v okvirju. MSS = MTU - 40(velikost headerjev) v tem primeru.

- negativni dogodki:
    - potek časovne kontrole: cwnd <- 1, od začetka
    - 3x podvojeni ACK -> prehod v fazo hitre obnove (fast recovery), v katero preideta počasen začetek in izogibanje zasičenju ob prejemu 3 ponovljenih ACK
        - namen: zapolnitev vrzeli, nato vrnitev v počasen začetek ali izogibanje zasičenju
        - nastavitev:
            - PRAG <- cwnd/2
            - cwnd <- cwnd/2 + 3

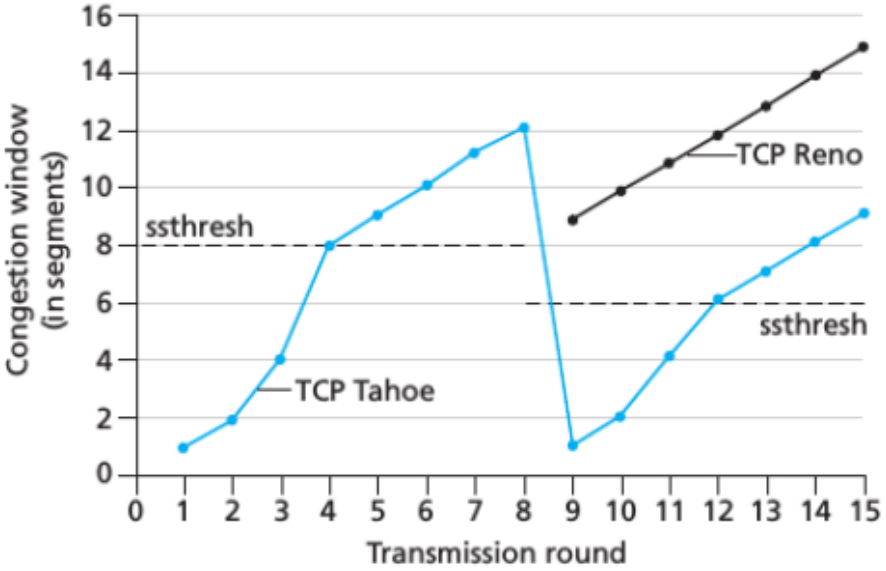


Končni avtomat za TCP nadzor zasičenja



Razvoj nadzora zasičenja skozi različice TCP

1. **TCP Tahoe:** osnovna verzija (uporablja samo počasen začetek in izogibanje zasičenja), po izgubi paketa vedno nastavi cwnd=1
2. **TCP Reno:** dodana faza hitre obnove - po prejemu treh kopij iste potrditve, preskoči počasen začetek in nastavi cwnd <- cwnd/2 + 3
3. **TCP Vegas:** dodano zaznavanje situacij, ki vodijo v zasičenje in linearno zmanjševanje hitrosti pošiljanja ob zasičenju



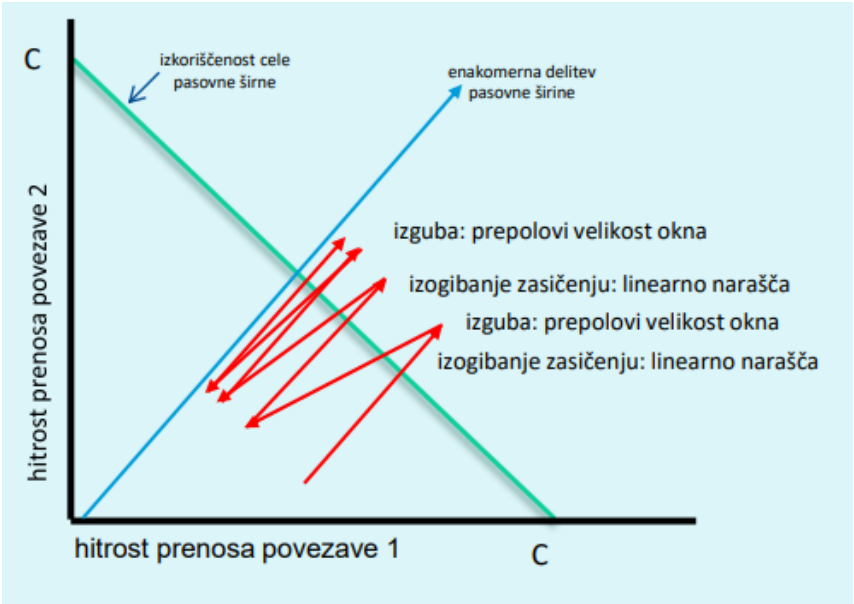
Primeri vprašanja za kolokvij/izpit

Primeri vprašanj za analizo delovanja protokola:

1. Kdaj se je izvajal počasen začetek in kdaj izogibanje zasičenju?
2. Kdaj so bile prejete 3 kopije iste potrditve in kdaj je potekel časovni interval?
3. Kakšen je bil prag na začetku, kakšen ob T=18, in T=24?
4. Če bi pri T=26 imeli 3 kopije potrditve, kako bi določili prag in cwnd?

Je TCP pravičen?

- **Cilj pravičnosti:** Vsaka od N TCP sej po isti povezavi s kapaciteto C naj bi dobila delež prenosa C/N.
- Izkaže se, da si več TCP pošiljateljev v praksi pravično deli pasovno širino (mehanizem nadzora zasičenja skonvergira v sredinsko točko grafa)



- Presečišče modre in zelene črte je ideal, kjer so vse seje pravične.
- Navzgor in desno od zelene črte prihaja do izgub saj je skupna hitrost pošiljanja vseh sej večja kot lahko kanal sprejme.

Pravičnost TCP in UDP

Sobivanje TCP in TCP v istem omrežju konvergira v pravično delitev.

- UDP in TCP po istem omrežju: ni pravično do TCP
    - UDP pošilja brez omejitev pretoka in se pri tem ne ozira na TCP.

Primer: če prižgemo torrent, ki se poveže z veliko semeni(veliko UDP povezav), je praktično nemogoče hkrati brskati po spletu(hitrost TCP sej se močno zmanjša saj UDP ni pravičen in ne zmanjšuje svoje hitrosti).