Računalniške komunikacije

2020/21

transportna plast

življenjski cikel
časovna kontrola
nadzor pretoka
nadzor številčenja
pravičnost TCP in UDP

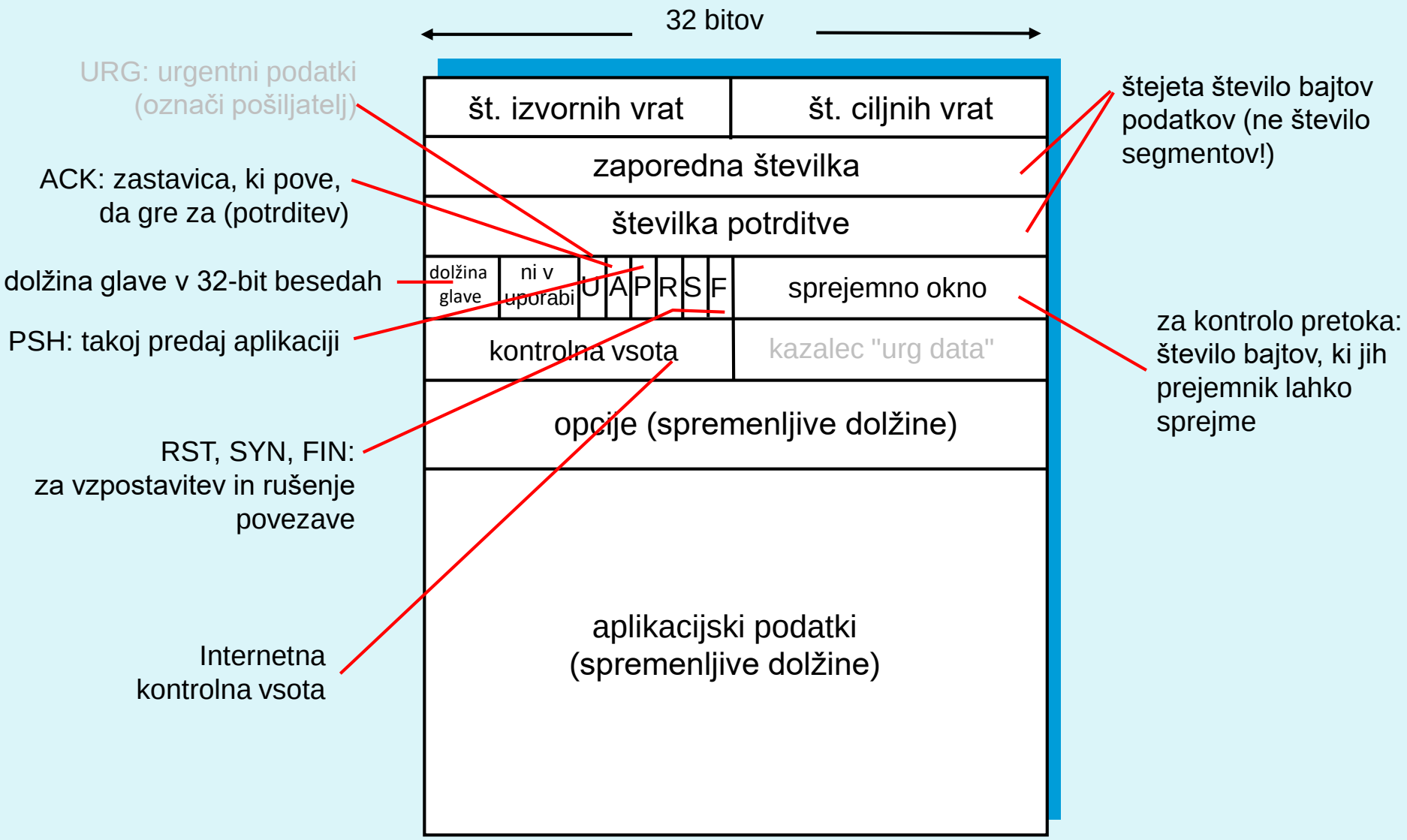
Pridobljeno znanje s prejšnjih predavanj

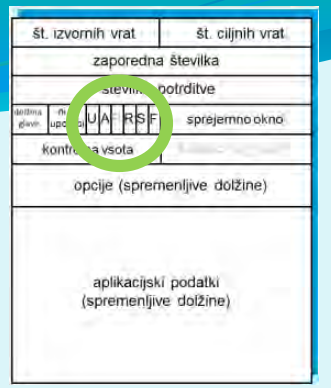
- **konstrukcija protokola TCP s končnimi avtomati**
  - posredno in kumulativno potrjevanje (samo ACKn za vse pakete $\leq$n) namesto ACK/NAK

- **tekoče pošiljanje** namesto **sprotnega potrjevanja**
  - ponavljanje N nepotrjenih (go-back-N)
  - ponavljanje izbranih (selective repeat)

- **protokol TCP**
  - številčenje segmentov, številčenje potrditev
  - vzpostavljanje povezave (SYN), trosmerno rokovanje
  - rušenje povezave (FIN)
  - nastavitev časovne kontrole
  - potrjevanje TCP
  - zakasnjeno potrjevanje
  - hitro ponovno pošiljanje (fast retransmit)
  - kontrola pretoka
  - kontrola zasičenja
  - pravičnost TCP in UDP

TCP segment


32 bitov
URG: urgentni podatki (označi pošiljatelj)
ACK: zastavica, ki pove, da gre za (potrditev)
dolžina glave v 32-bit besedah
PSH: takoj predaj aplikaciji
RST, SYN, FIN: za vzpostavitev in rušenje povezave
Internetna kontrolna vsota
št. izvornih vrat
št. ciljnih vrat
zaporedna številka
številka potrditve
dolžina glave
ni v uporabi
U
A
P
R
S
F
sprejemno okno
kontrolna vsota
kazalec "urg data"
opcije (spremenljive dolžine)
aplikacijski podatki (spremenljive dolžine)
štejeta število bajtov podatkov (ne število segmentov!)
za kontrolo pretoka: število bajtov, ki jih prejemnik lahko sprejme

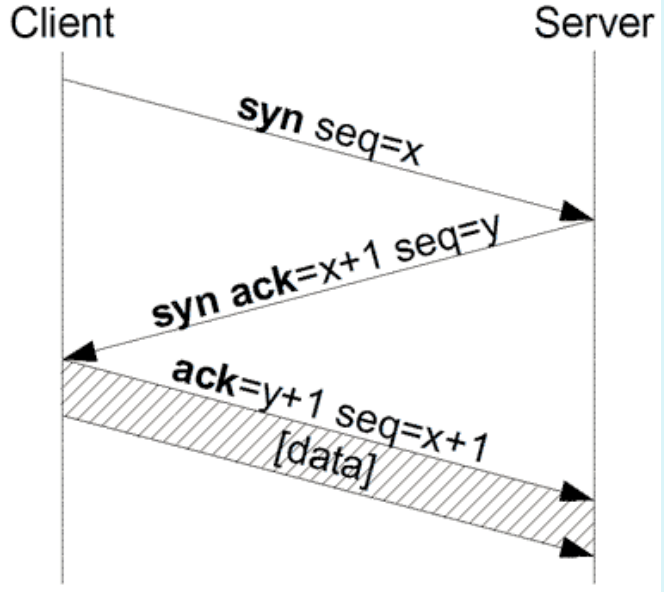


TCP: vzpostavljanje povezave

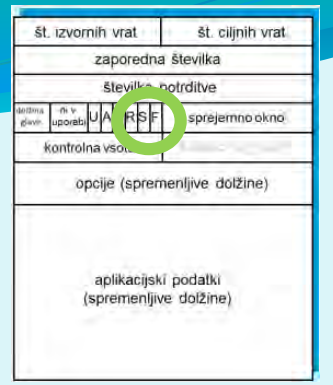
- pošiljatelj in prejemnik pred pošiljanjem izvedeta rokovanje (handshake), v katerem izmenjata parametre:
  - **začetne pričakovane zaporedne številke** (naključno določene)
  - **velikosti medpomnilnikov** (za kontrolo pretoka)

- trojno rokovanje (*three-way handshake*)

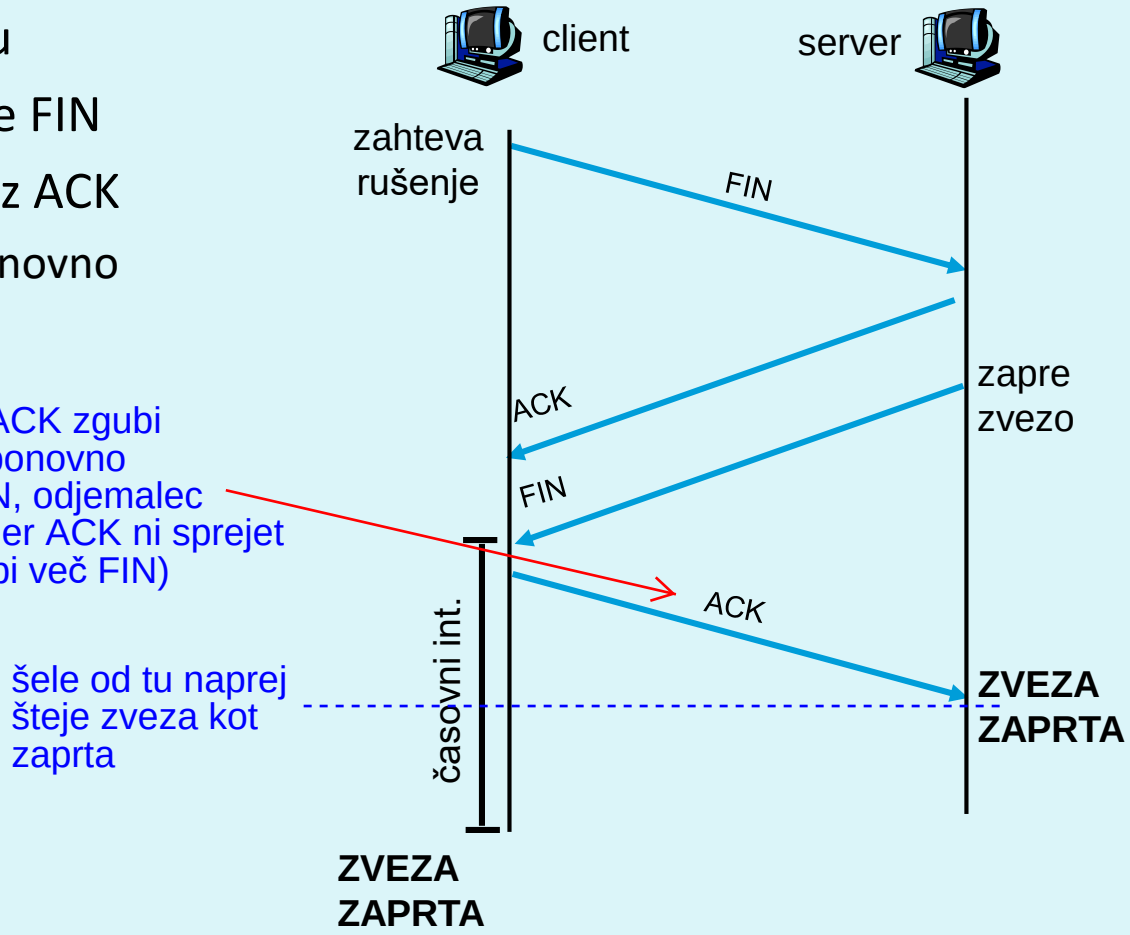
  1. Odjemalec pošlje segment z zastavico **SYN** (sporoči začetno številko segmenta, ni podatkov)
  2. Strežnik vrne segment **SYN ACK** (rezervira medpomnilnik, odgovori z začetno številko svojega segmenta)
  3. Odjemalec vrne **ACK**, lahko že s podatki ("štuporama")



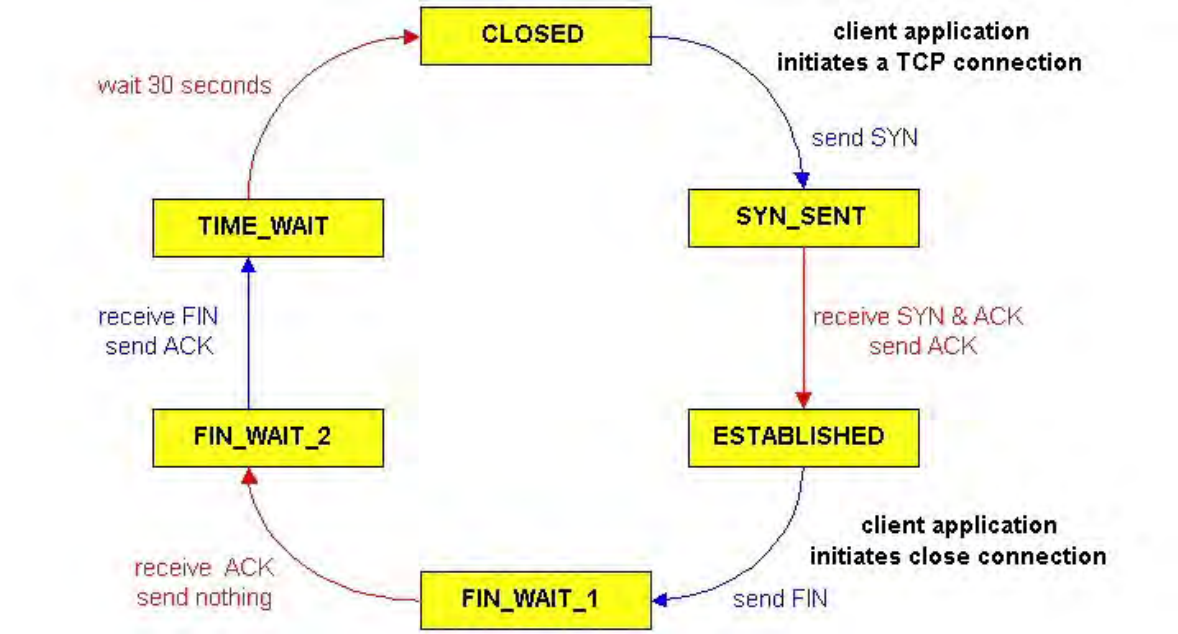
TCP: rušenje povezave



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, končano

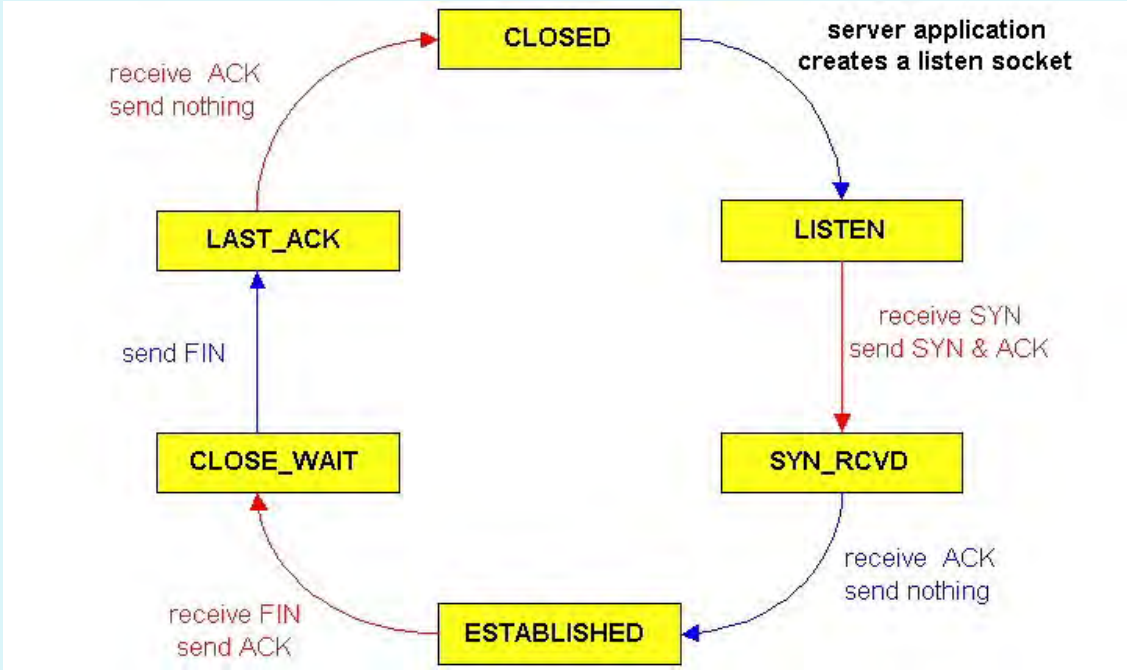


Življenjska cikla odjemalca in strežnika



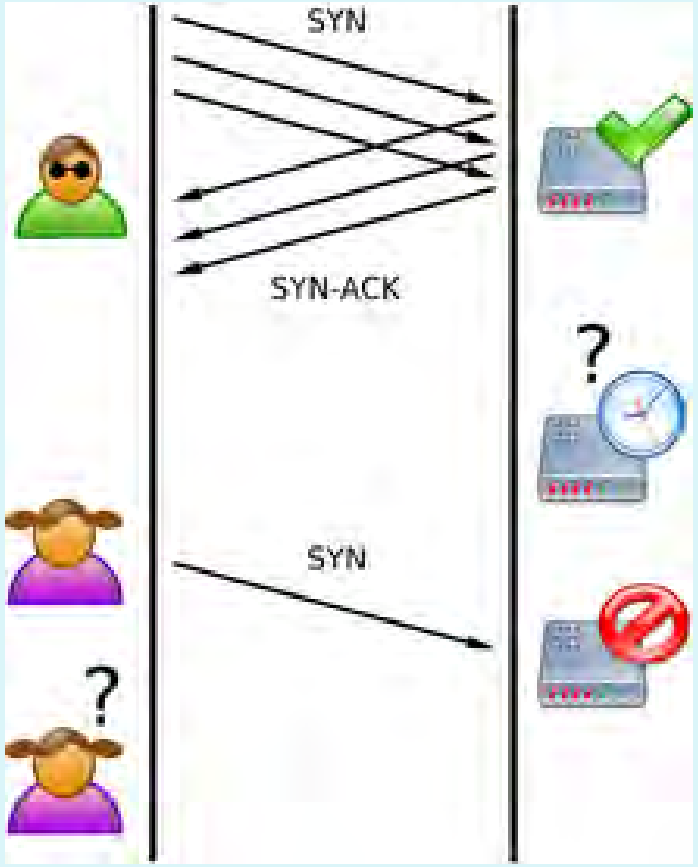
TCP odjemalec

TCP strežnik



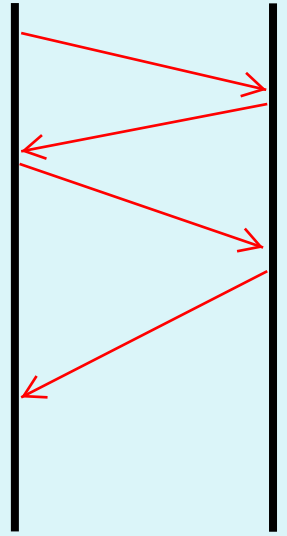
Varnost: napad SYN FLOOD

- 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*)
  - porazdeljeni DoS napad: pošiljanje TCP SYN iz več virov



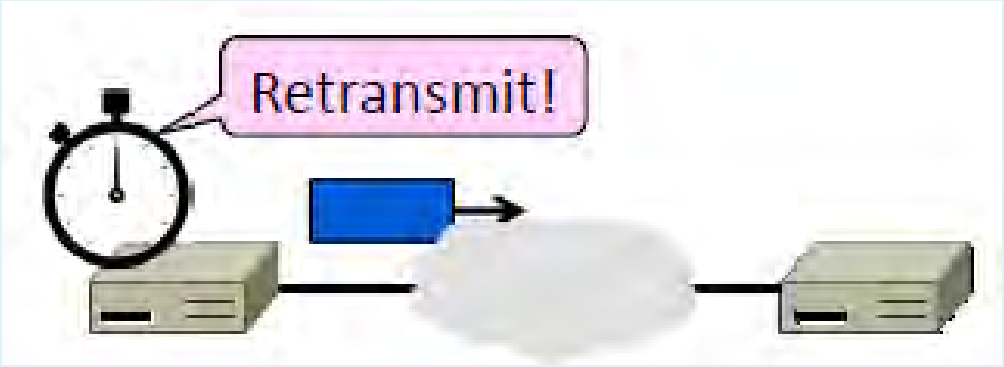
Nastavitev časovne kontrole

RTT ni vedno enak,
odvisno od stanja omrežja



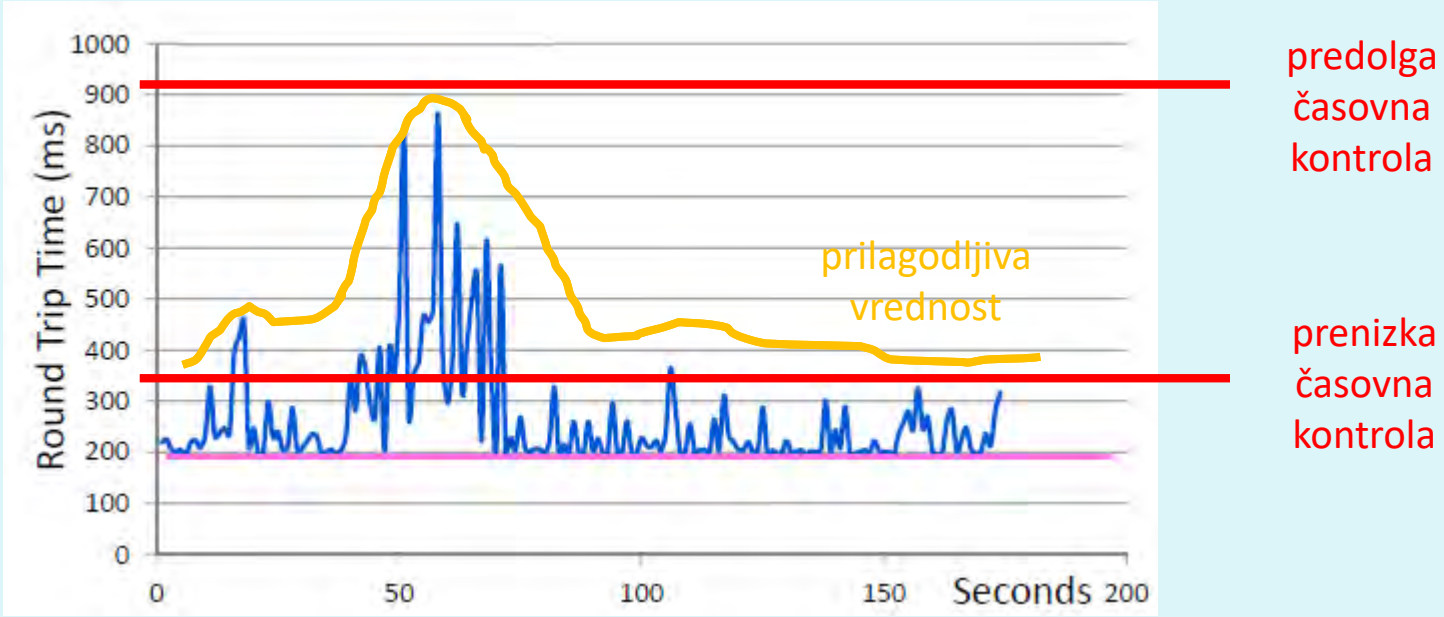
- **časovna kontrola (štoparica)**: potrebna za uporabo zanesljive dostave, t. j. ponovnega pošiljanja, če se izgubi paket ali njegova potrditev

- 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

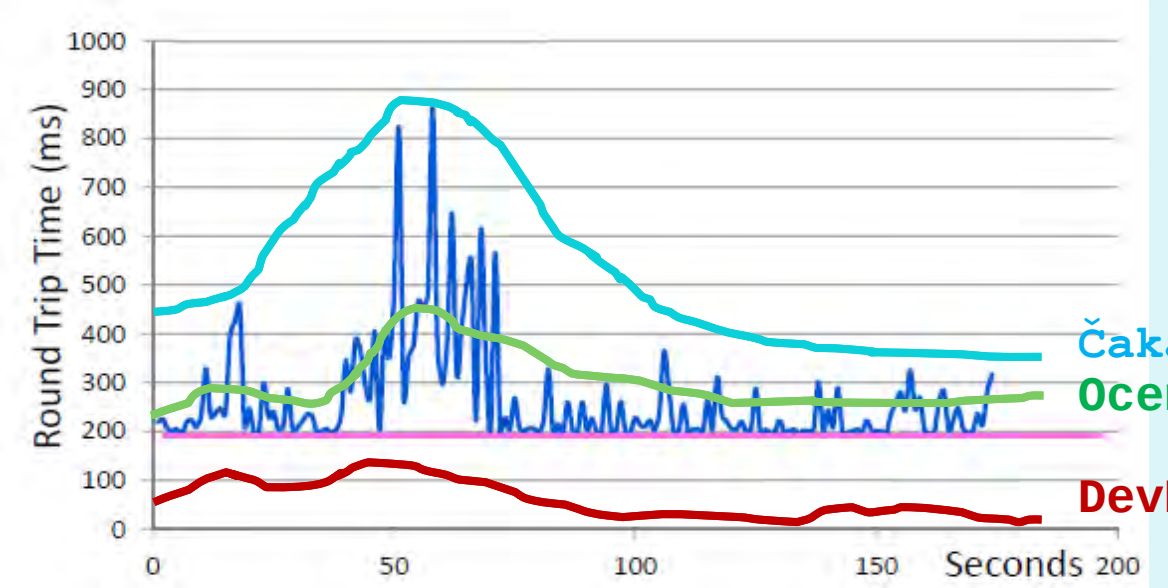


Primer ocenjevanja RTT

- 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



Primer ocenjevanja RTT



- izračunamo gibajoče povprečje
  `OcenjeniRTT[i] <- (1-α)*OcenjeniRTT[i-1] + α*IzmerjeniRTT[i]`
  običajno uporabimo: α=0.125

  prejšnja vrednost zelene krivulje

  trenutna vrednost modre krivulje

- izračunamo gibajočo deviacijo
  `DevRTT[i] = (1-β)*DevRTT[i-1] + β*|IzmerjeniRTT[i] - OcenjeniRTT[i]|`
  običajno uporabimo β=0.25

  odstopanje med zeleno krivuljo in modro krivuljo

- vrednost čakalnega intervala TCP nastavi kot OcenjeniRTT + "rezerva":
  `ČakalniInterval[i] = OcenjeniRTT[i] + 4*DevRTT[i]`

Način potrjevanja

- katere vrste tekočega potrjevanja uporablja TCP?
  - ponavljanje N nepotrjenih (*go-back-N*)?
  - potrjevanje posameznih (*selective repeat*)?

- ODGOVOR: uporablja **kombinirano rešitev** obeh
  - podoben ponavljanju N nepotrjenih (štoparica za **najstarejši nepotrjeni segment**), vendar ob poteku časovne kontrole ne pošlje vseh segmentov v oknu, temveč **le najstarejši nepotrjeni** segment
  - RFC2018 vpeljuje potrjevanje le izbranih paketov

Posebnosti pri potrjevanju TCP

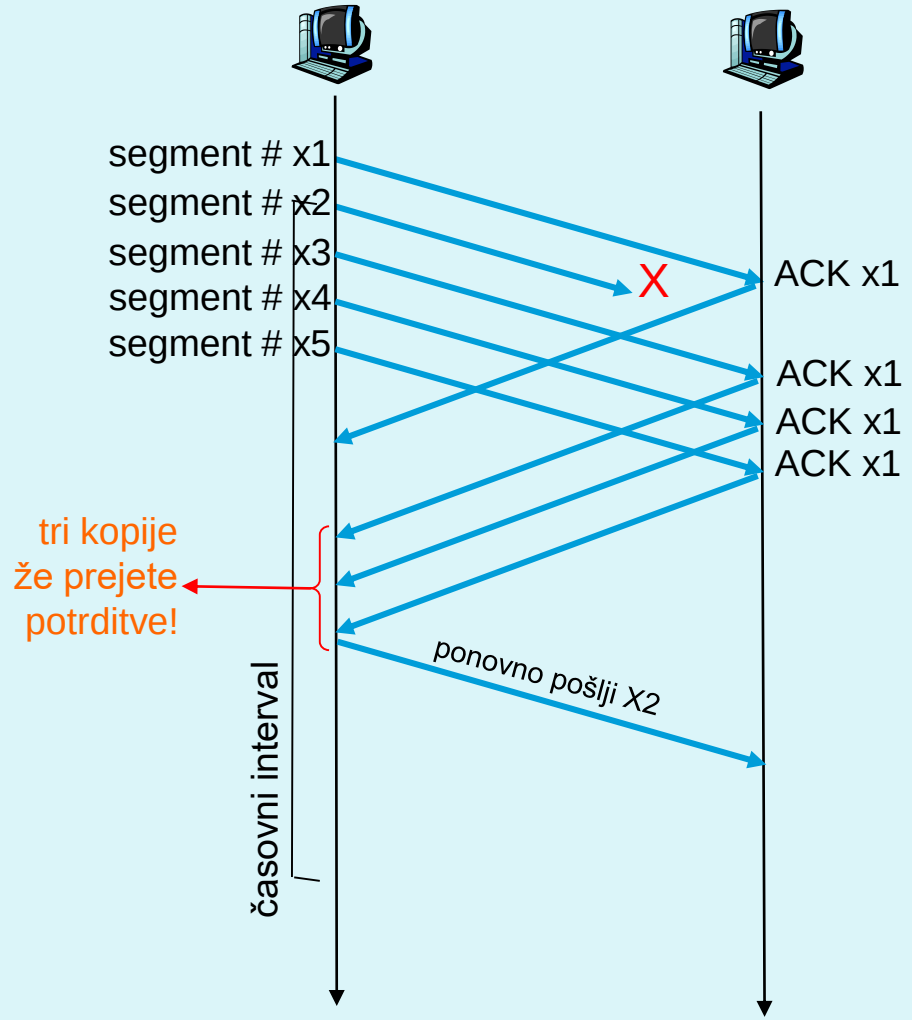
Dogodek pri prejemniku	Odziv prejemnika
Sprejem segmenta s pričakovano številko, vsi prejšnji že potrjeni.	Počakaj na naslednji segment max 500 ms. Če ta pride v tem intervalu, izvedi **zakasnjeno potrditev obeh** (delayed ACK). Če ne pride v tem intervalu, potrdi samo prejetega.
Isto kot zgoraj, a potrditev 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.

to ni čakalna kontrola od prej

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** (*fast retransmit)* pošiljatelj izvede **pred potekom časovnega intervala**, če prejme za nek paket 3 podvojene potrditve

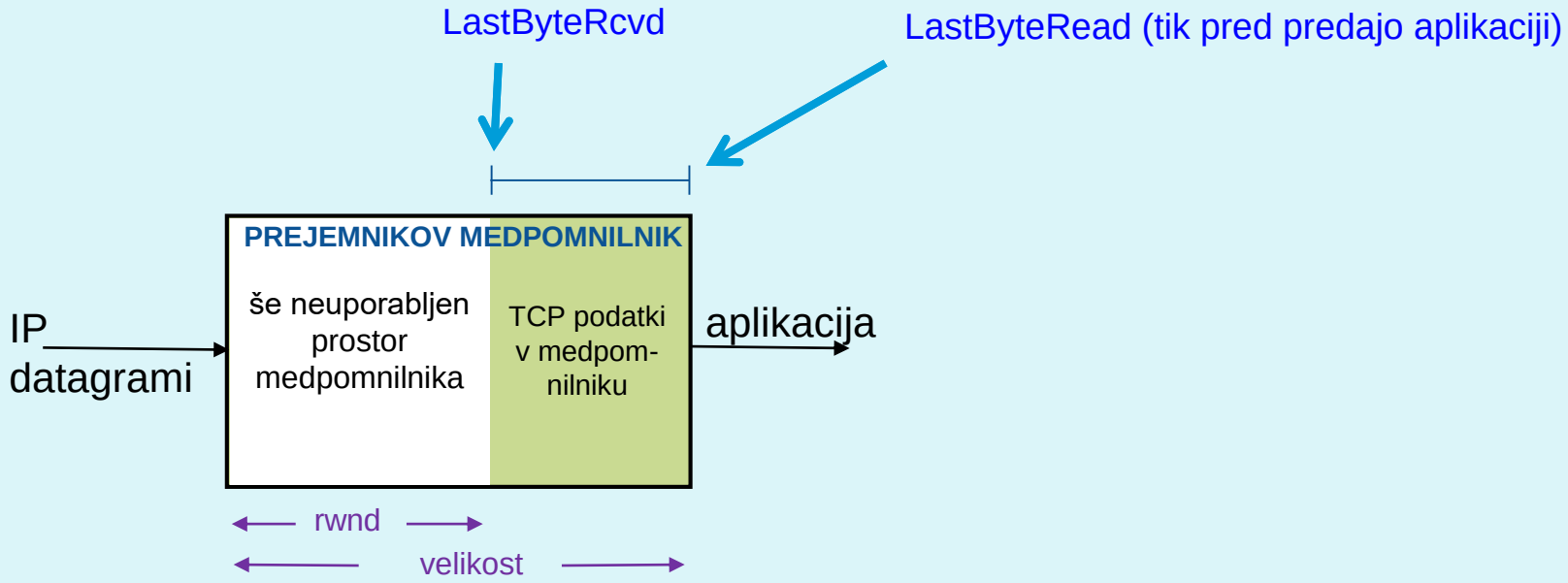


Kontrola pretoka TCP

- 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:

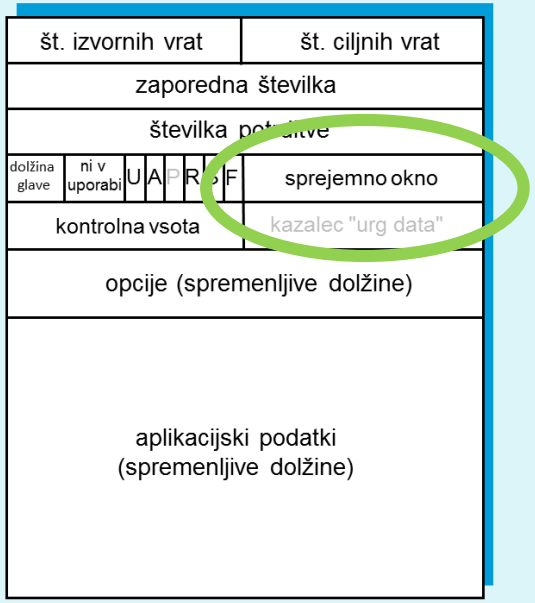
```
rwnd = velikost - [LastByteRcvd - LastByteRead]
```

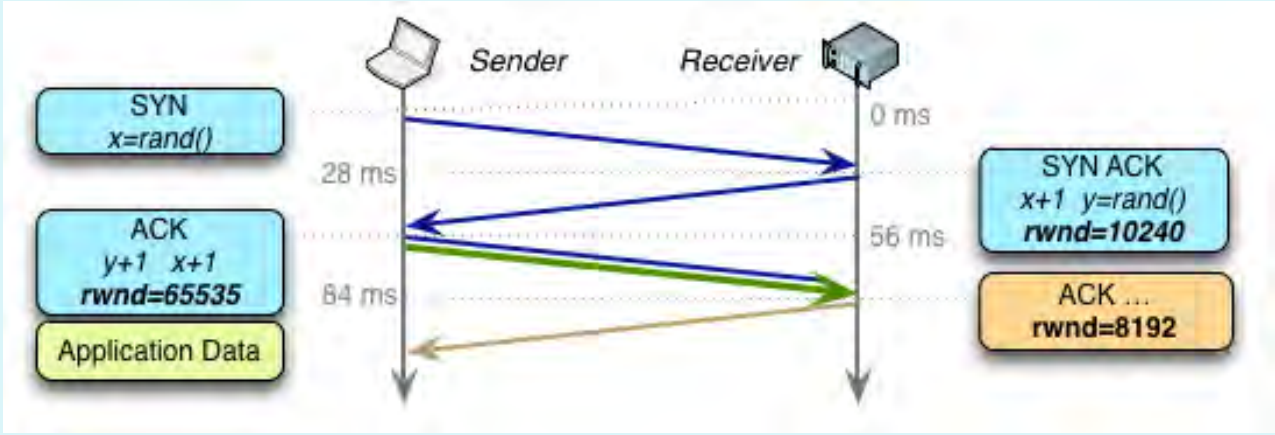
rwnd = receive window



Kontrola pretoka TCP

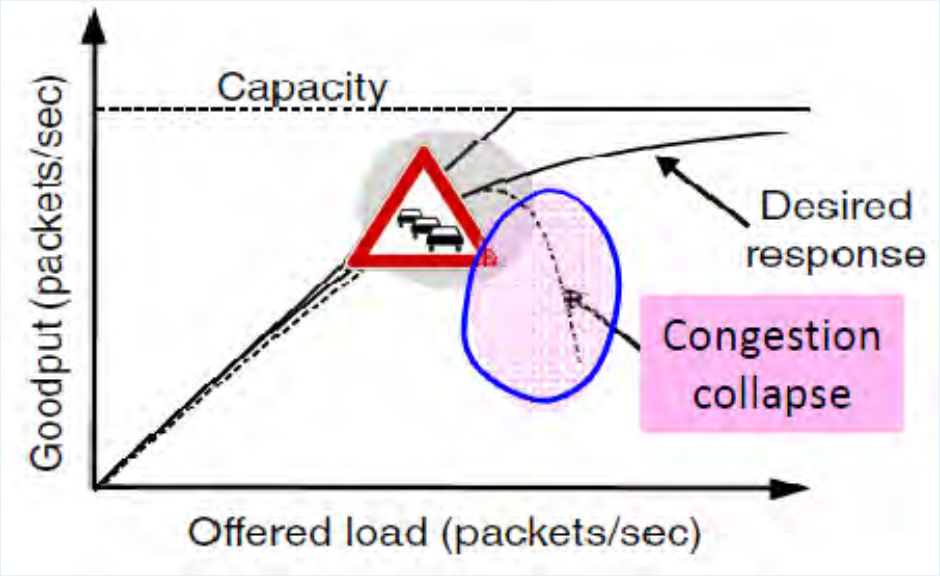
- prejemnik sporoča pošiljatelju velikost razpoložljivega prostora v glavi vsakega segmenta (rwnd)
- pošiljatelj ustrezno omeji število paketov, za katere še ni prejel potrditve





Nadzor zasičenja

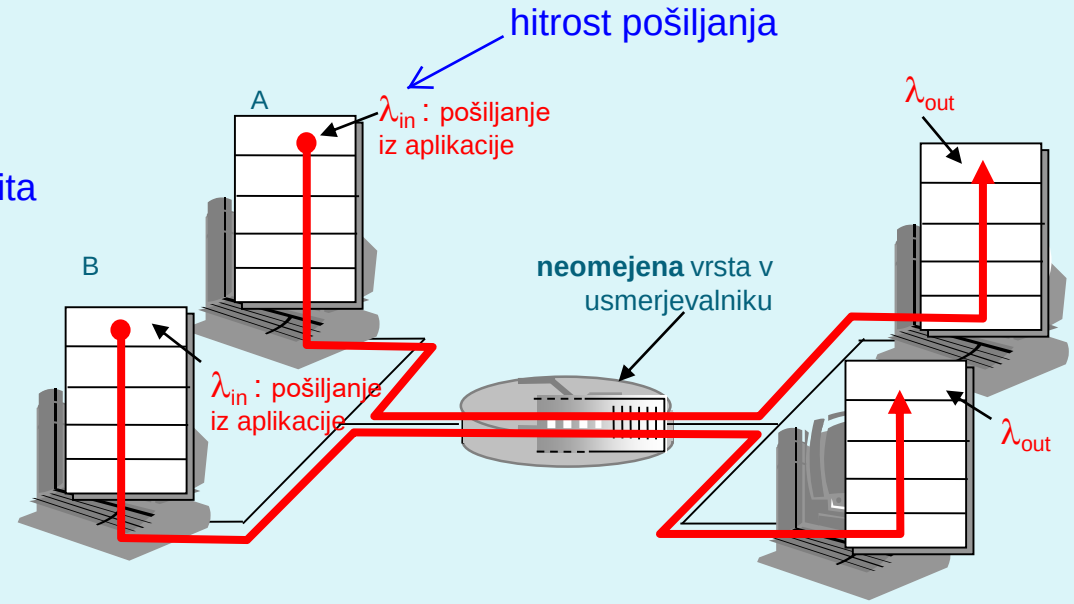
- zasičenje: stanje omrežja, ko veliko virov naenkrat prehitro pošilja preveč podatkov za dano omrežje
- posledica zasičenja:
  - izguba segmentov (prekoračitve medpomnilnika v usmerjevalnikih)
  - velike zakasnitve (čakalne vrste v usmerjevalnikih)
- ni isto kot nadzor pretoka!

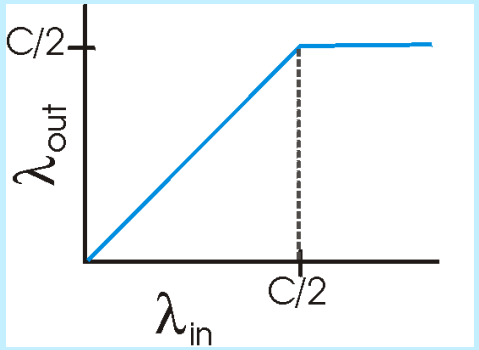


Zasičenje – primer 1

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

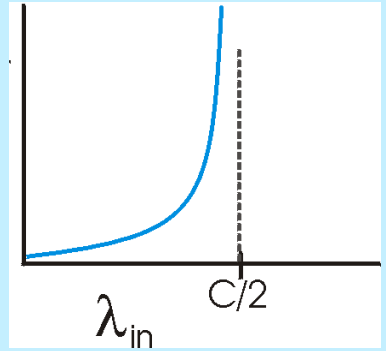
A in B si pravično razdelita kanal, vsak dobi C/2





pretok:

- s stališča pretoka je idealno pošiljanje s hitrostjo C/2



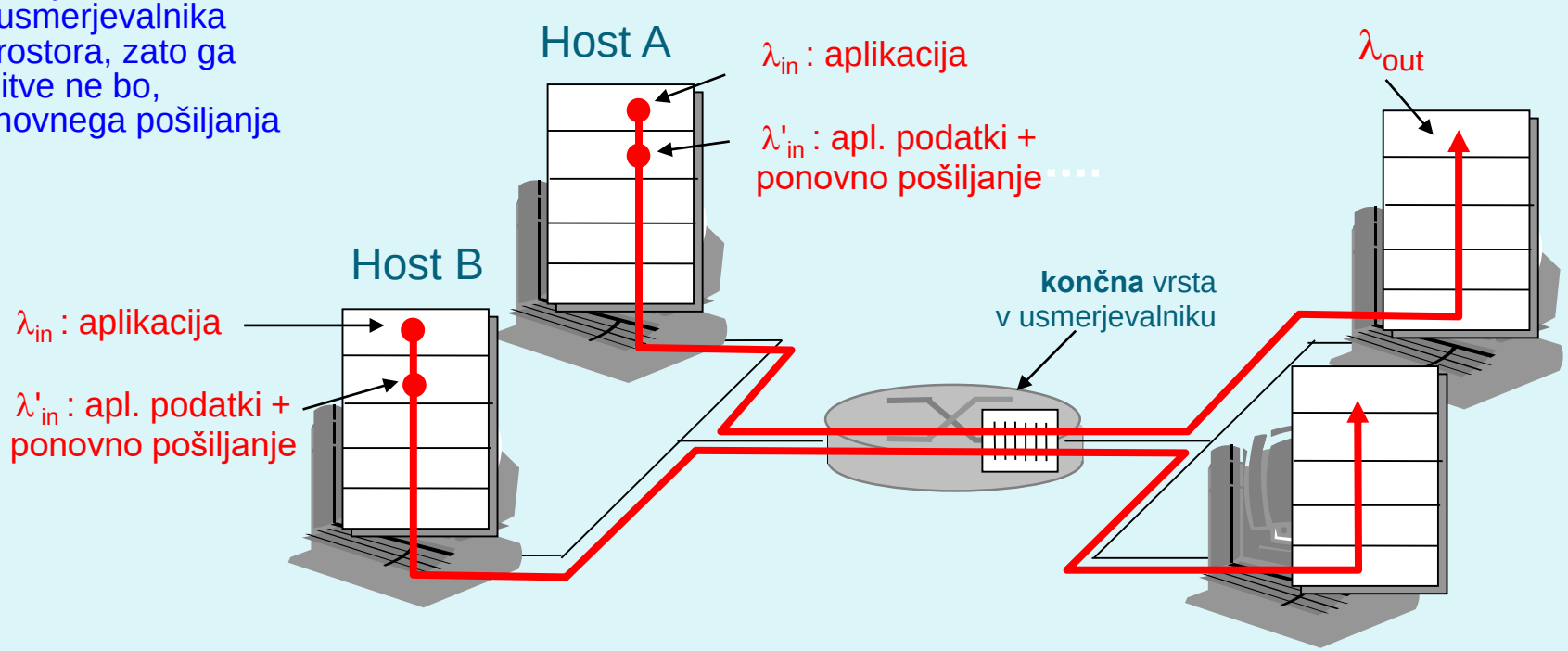
zakasnitev:

- s stališča zakasnitve pošiljanje z večjo hitrostjo polni čakalno vrsto v neskončnost

Zasičenje – primer 2

- končna vrsta
- ponovna pošiljanja segmentov zaradi izgub (vrste) in zakasnitev

če A in B pošiljata prehitro,
paket pride do usmerjevalnika
in zanj ni več prostora, zato ga
zavrže → potrditve ne bo,
prišlo bo do ponovnega pošiljanja



Zasičenje – primer 2

↓ EFEKTIVNOST ↓

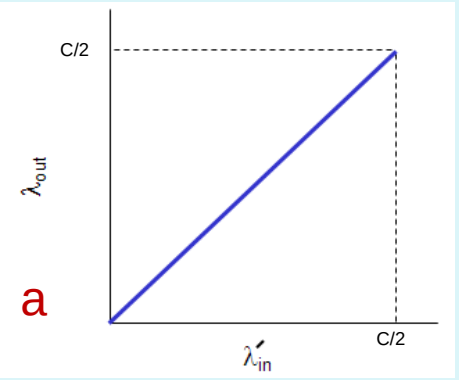
Preučimo tri scenarije:

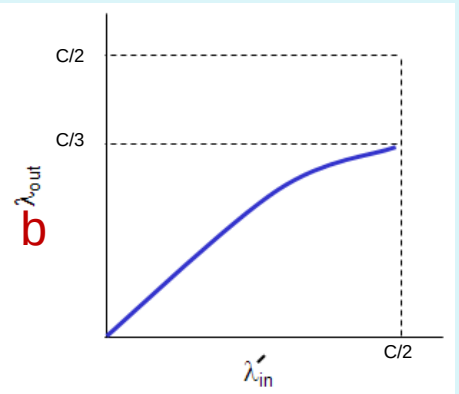
a. segment oddamo le, ko je prostor v vrsti, tako da ni izgub (v praksi to ni možno, ker tega ne vemo)

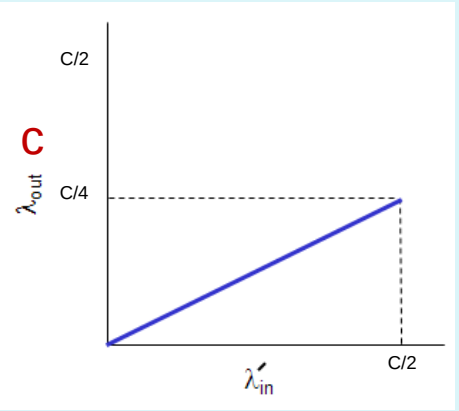
b. dogajajo se izgube paketov in ponovna pošiljanja

c. ponovna pošiljanja tudi zaradi velikih zakasnitev

Torej: Več dela omrežja za manjši učinek. Nepotrebne ponovitve.

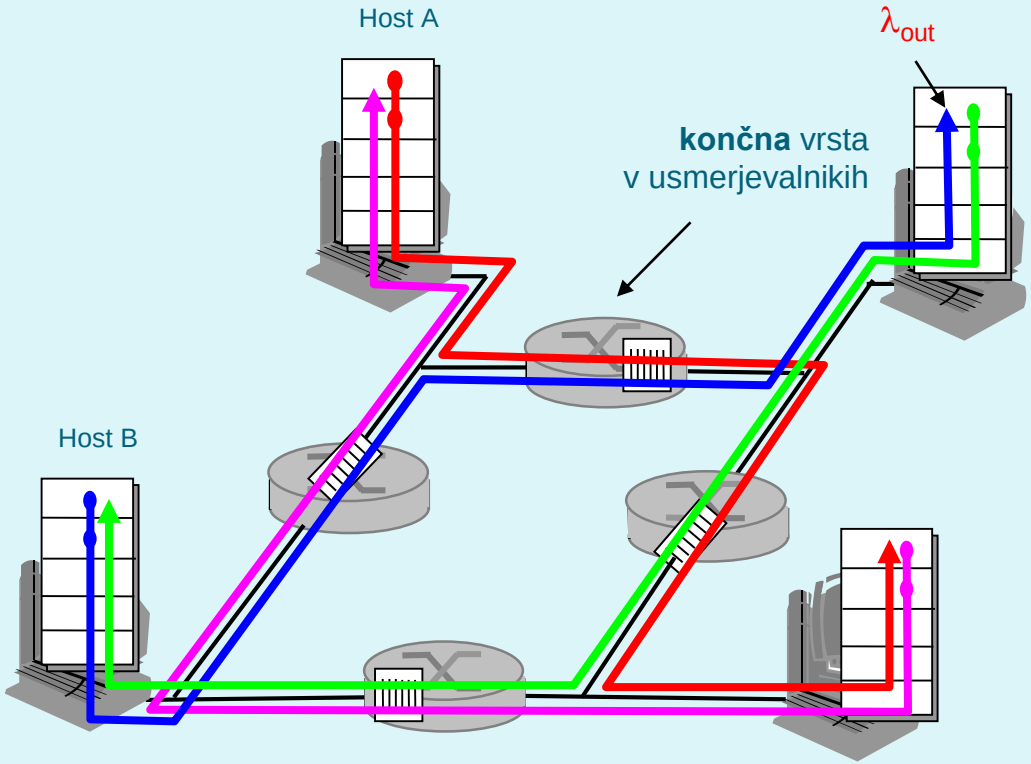


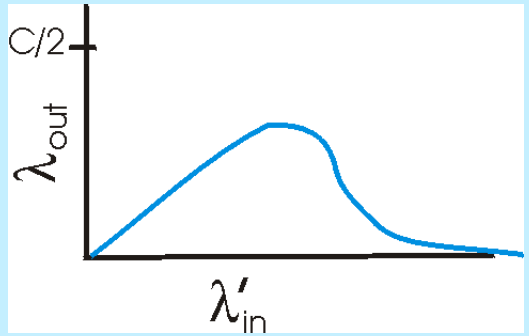




Zasičenje – primer 3

- daljše poti: če se paket izgubi na *n*-tem skoku, so bili zaman vsi dotedanji prenosi!





pošiljanje z večjo hitrostjo polni čakalno vrsto v neskončnost

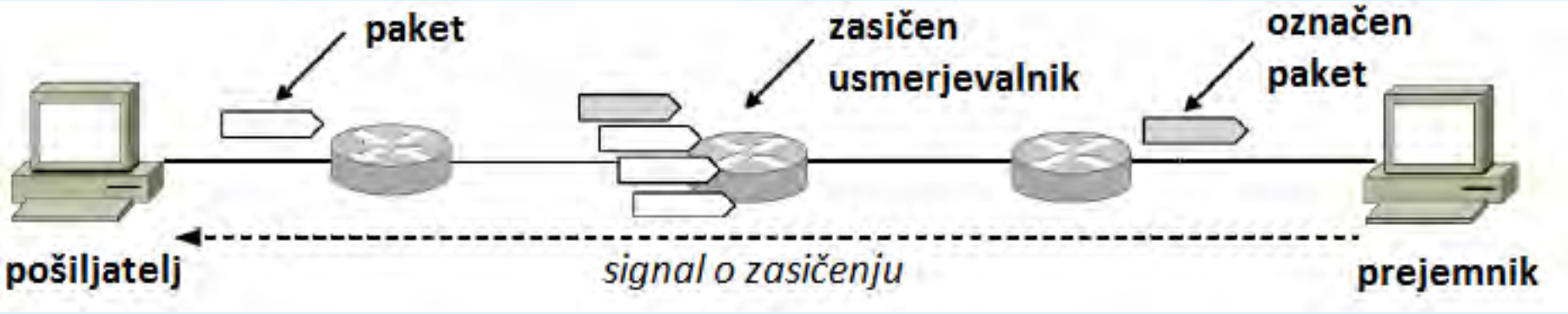
Kako nadzorovati zasičenja?

usmerjevalnikova vrsta se polni

dva pristopa:

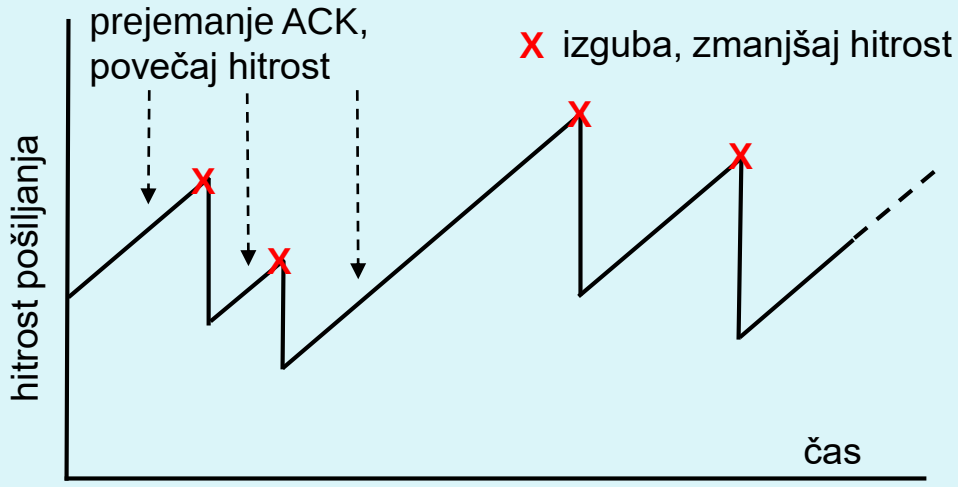
1. z uporabo **omrežnih storitev**: usmerjevalniki v omrežju obvestijo pošiljatelja, da je prišlo do zasičenja
   - uporaba obvestila o zasičenju (ECN – explicit congestion notification): usmerjevalnik nastavi ustrezen bit in sporoči sprejemljivo hitrost oddajanja (npr. pri ATM)

2. 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

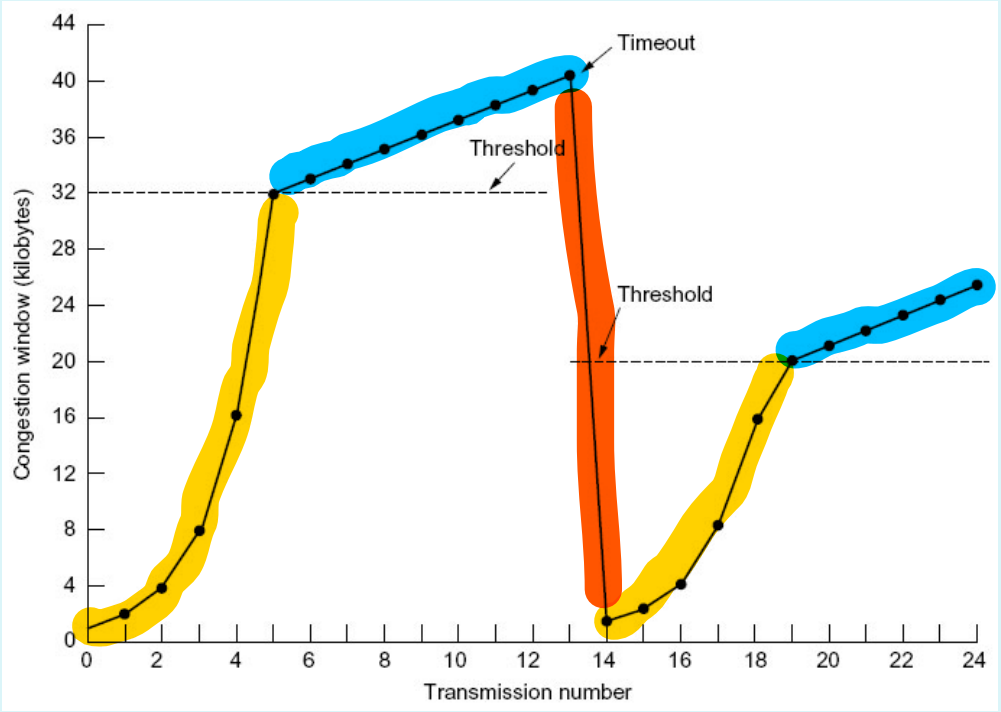
- **ideja**: pošiljatelj želi pošiljati ČIM HITREJE, vendar še POD MEJO zasičenja omrežja. Kako najti pravo hitrost?
- **rešitev**: vsak pošiljatelj si sproti nastavlja hitrost na podlagi opazovanja reakcij v omrežju na pošiljanje:
  - če prejme potrditev (ACK), ni zasičenja, poveča hitrost
  - če se segment izgubi, je to posledica zasičenja, zmanjšaj hitrost



TCP ima "žagasto obliko" hitrosti pošiljanja

TCP nadzor zasičenja

- okno **rwnd** (*receive window*) smo že spoznali (omejitev količine nepotrjenih podatkov za kontrolo pretoka)
- za nadzor zasičenja uporabljamo okno **cwnd** (*congestion window*). TCP torej pošilja s hitrostjo, ki ustreza **min(rwnd, cwnd)**
- **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

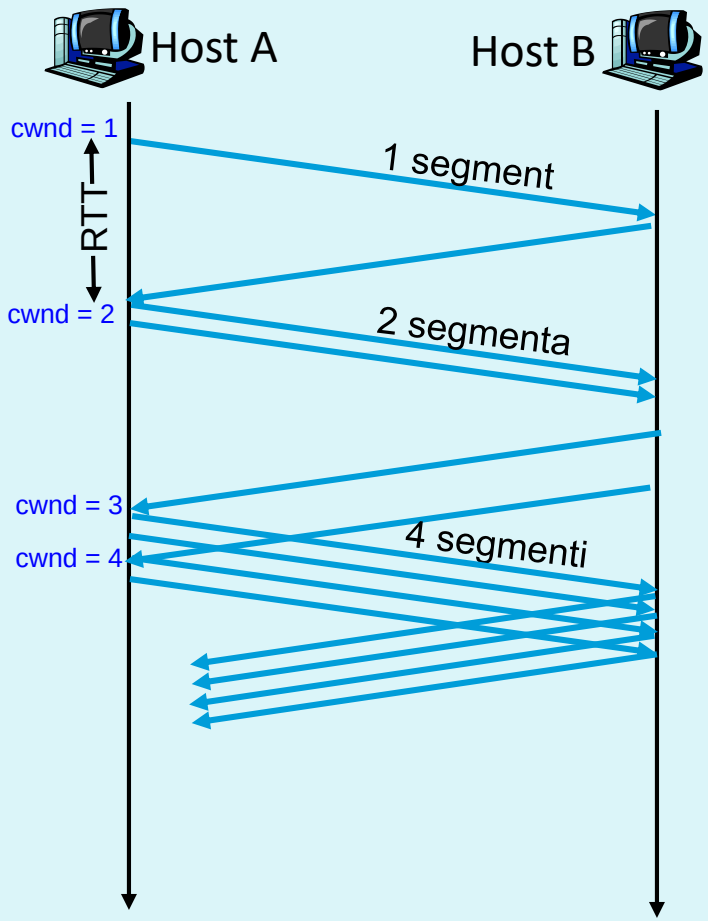


Počasen začetek (*Slow Start*)

cwnd ima enoto MSS (maximum segment size)

za VSAK prejeti ACK!!
tj. če dobimo 4, to pomeni
4 + 1 + 1 + 1 + 1 = 8

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**.



Izogibanje zasičenju (*Congestion Avoidance*)

če pride do izgube pri cwnd = 32, bo PRAG = 16

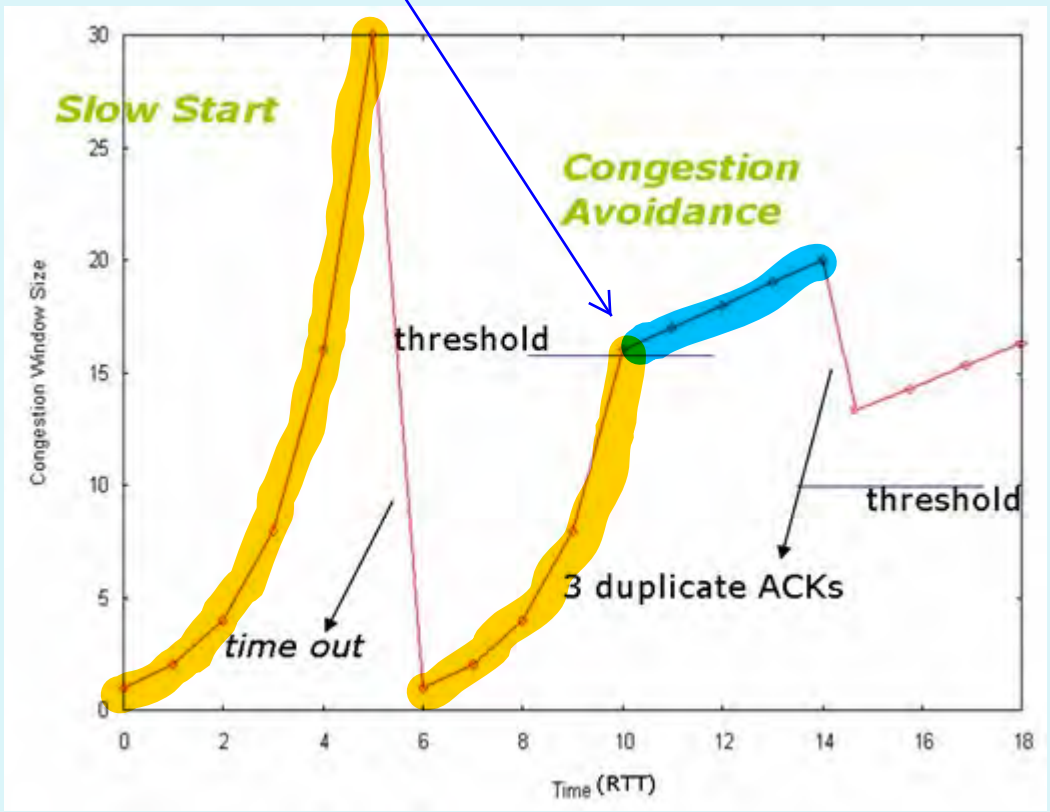
- kadar **cwnd > PRAG**, povečuj cwnd linearno za 1 MSS
- na ta način se bolj počasi približaj pragu zasičenja

v fazi linearnega povečevanja za VSE prejete ACK-e (tj. vsi se morajo vrniti) poveča cwnd za 1

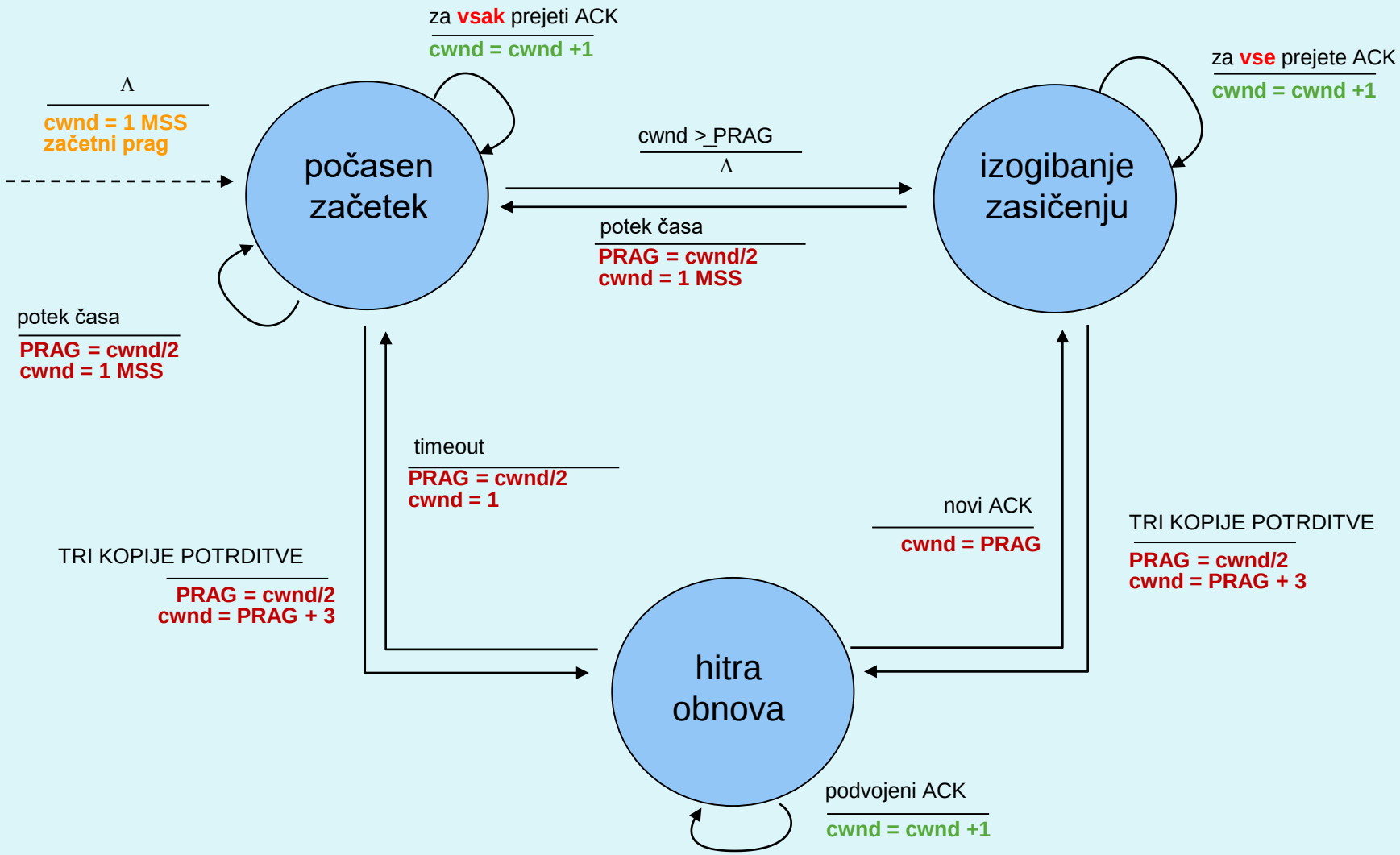
- negativni dogodki:
  - **potek časovne kontrole**: cwnd <- 1, od začetka
  - **3x podvojeni ACK** -> prehod v fazo **hitre obnove** (*fast recovery*), v katero preideta počasen začetek in izogibanje zasičenju ob prejemu 3 ponovljenih ACK
    - **namen**: zapolnitev vrzeli, nato vrnitev v počasen začetek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

ob izgubi samo enega, da hitrosti ne zmanjšamo preveč

tukaj ve od prej, da če gre čez PRAG, bo verjetno prišlo do težav (= izgub), zato gre v CA

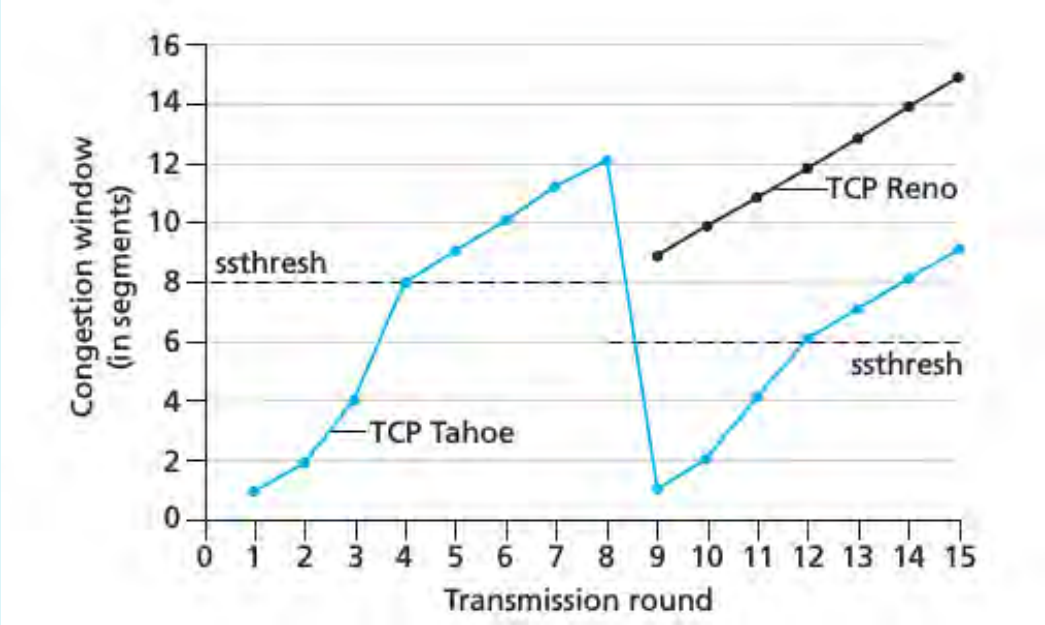


Končni avtomat za TCP nadzor zasičenja

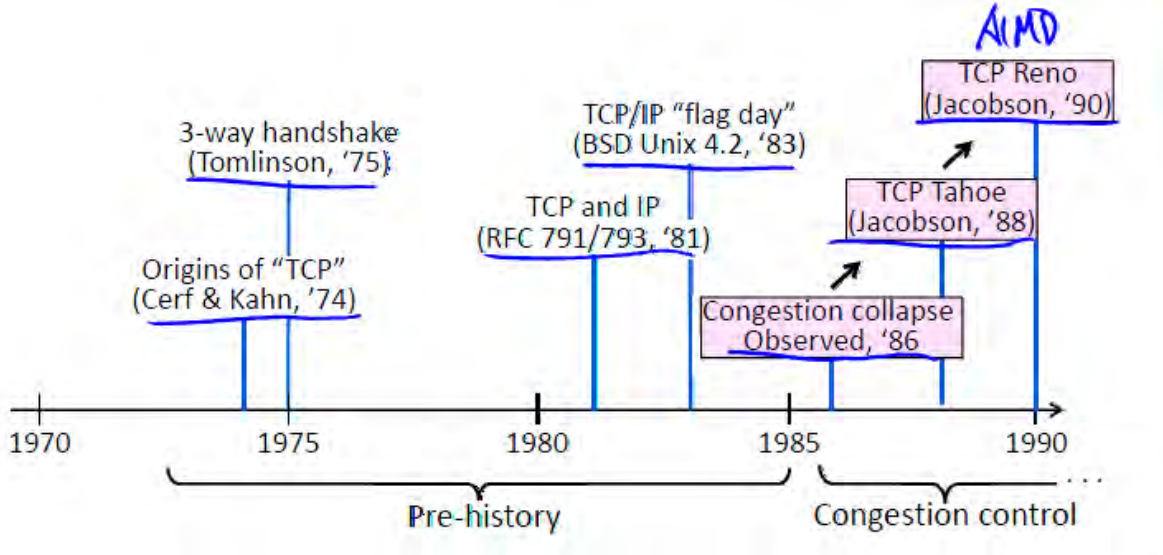


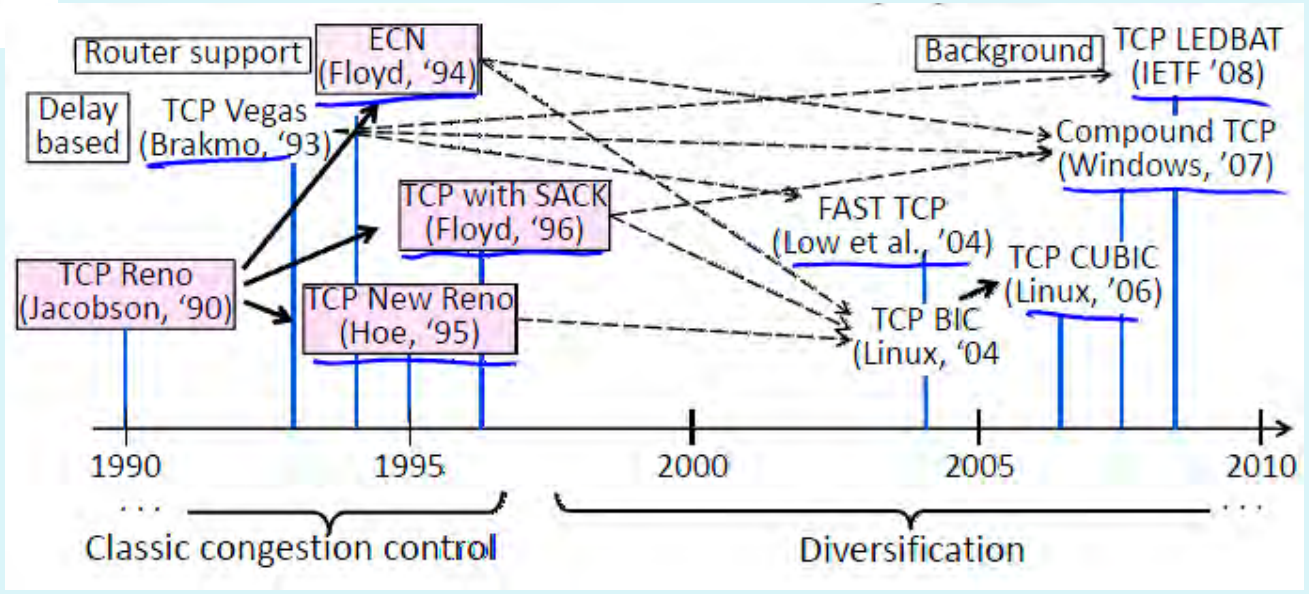
Razvoj nadzora zasičenja skozi različice TCP

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

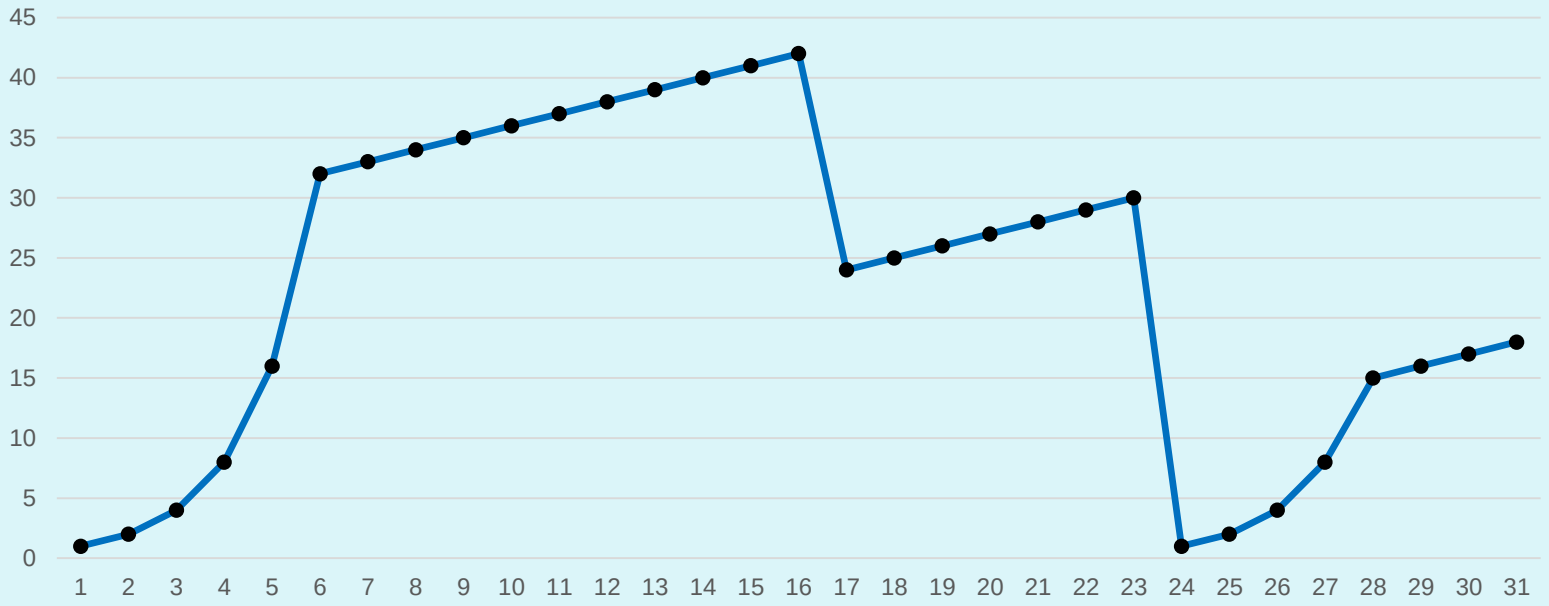


Zgodovina razvoja TCP





Primer: analiza delovanja TCP Reno



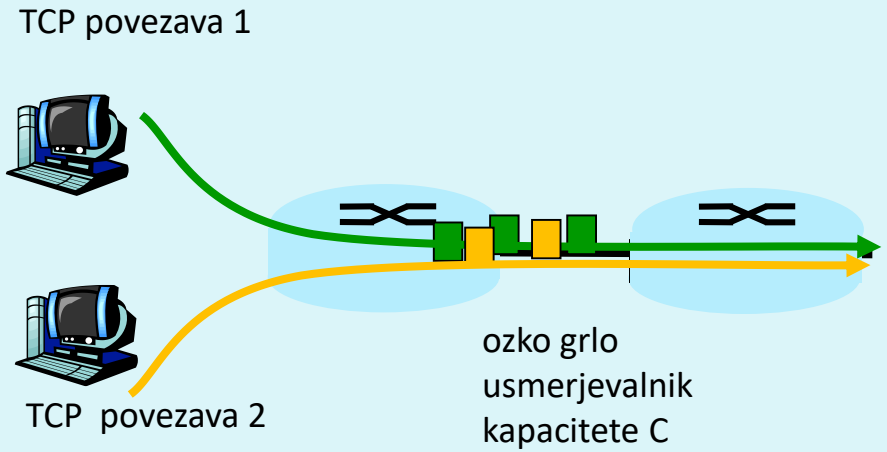
Primeri vprašanj za analizo delovanja protokola:

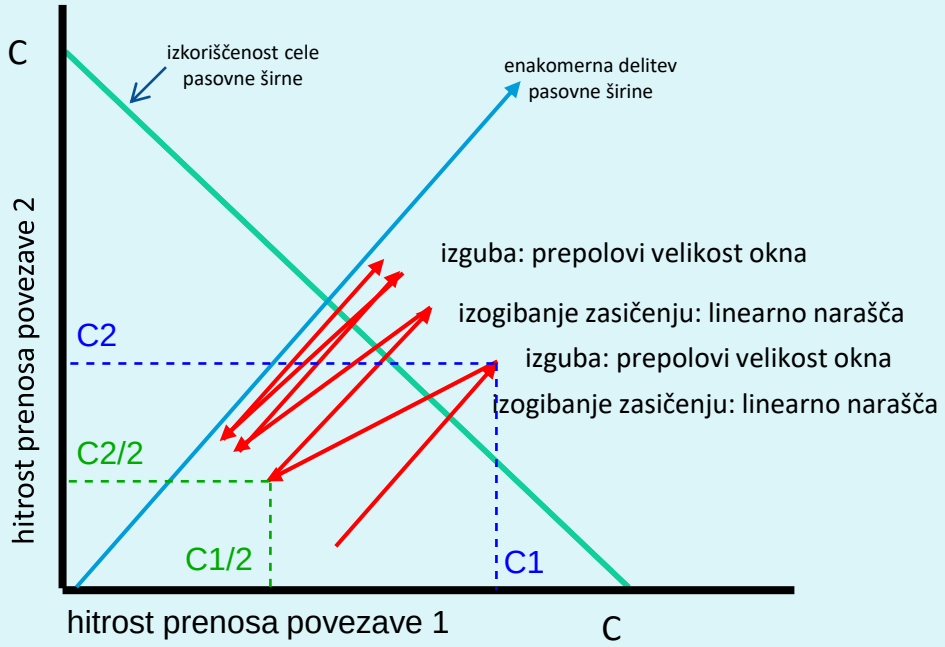
1. Kdaj se je izvajal počasen začetek in kdaj izogibanje zasičenju?
2. Kdaj so bile prejete 3 kopije iste potrditve in kdaj je potekel časovni interval?
3. Kakšen je bil prag na začetku, kakšen ob T=18, in T=24?
4. Če bi pri T=26 imeli 3 kopije potrditve, kako bi določili prag in cwnd?

Je TCP pravičen?

(Razlaga: predavanje 26. 04. 2021, cca 15-20 minut po začetku)

- 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)





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

Nadaljevanje?

- aplikacijska plast!