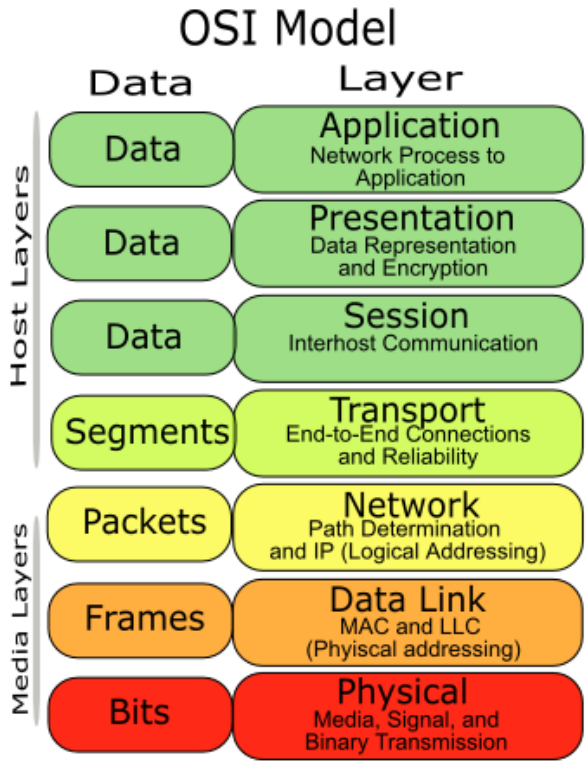
Transportna plast

UDP, TCP, potrjevanje

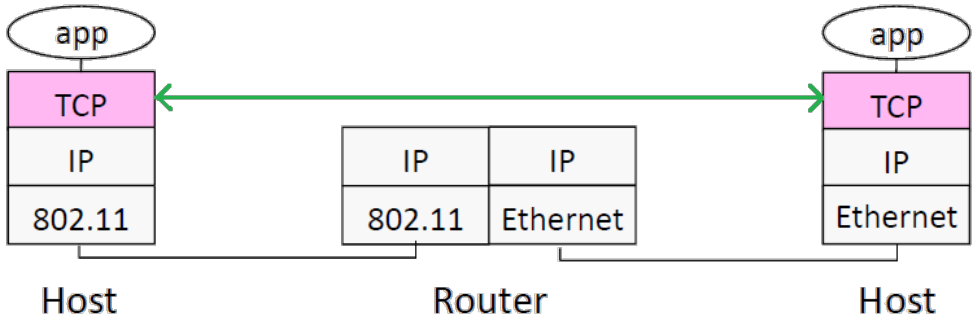
Mojca Ciglarič

Transportna plast



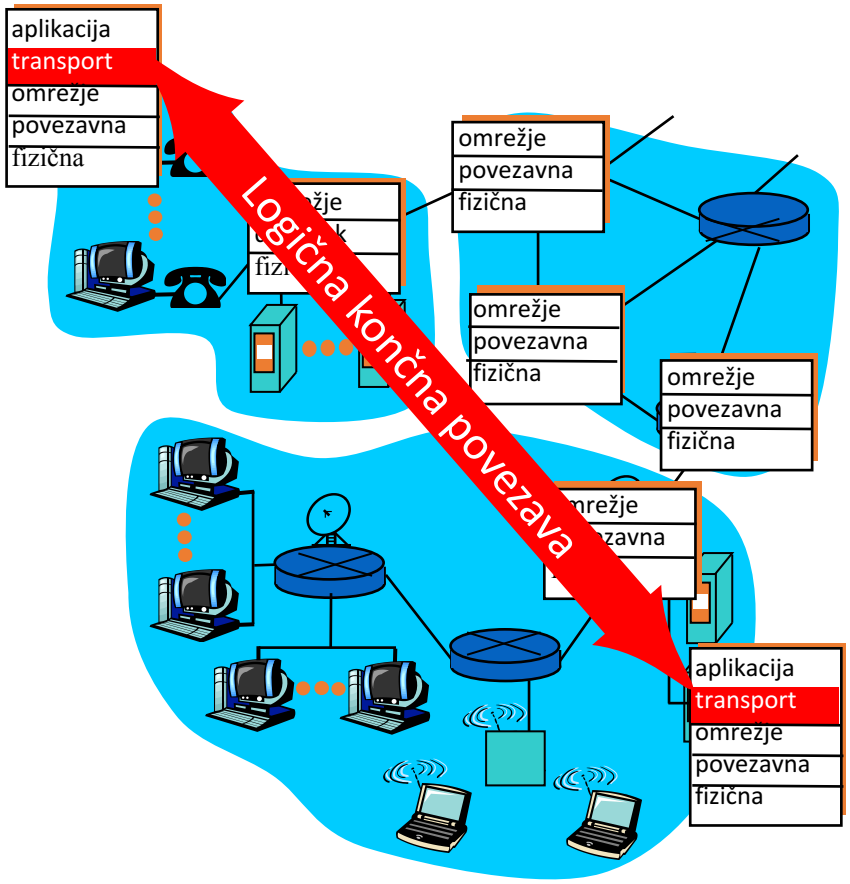
Naloge transportne plasti:

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



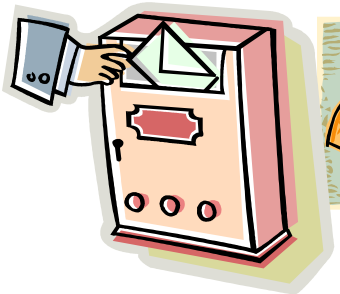
Storitve transportne plasti

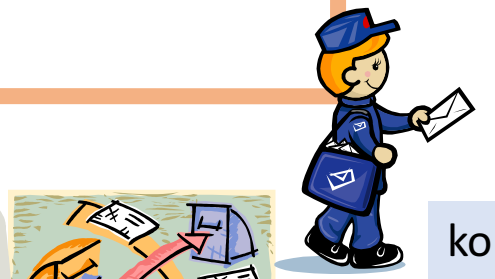
- Logična komunikacija med aplikacijskimi procesi
  - **pošiljatelj**: sporočilo razbije v SEGMENTE in jih posreduje v enkapsulacijo omrežni plasti
  - **prejemnik**: dekapsulira segmente iz paketov, sestavljene segmente združi v sporočila in jih posreduje aplikacijski plasti
- Protokola TCP in UDP



Analogija

Ana





Eva

končni sistem = hiša
proces = otrok
aplikacijsko sporočilo = pismo
transportni protokol = Ana in Eva
omrežna plast = storitev pošte

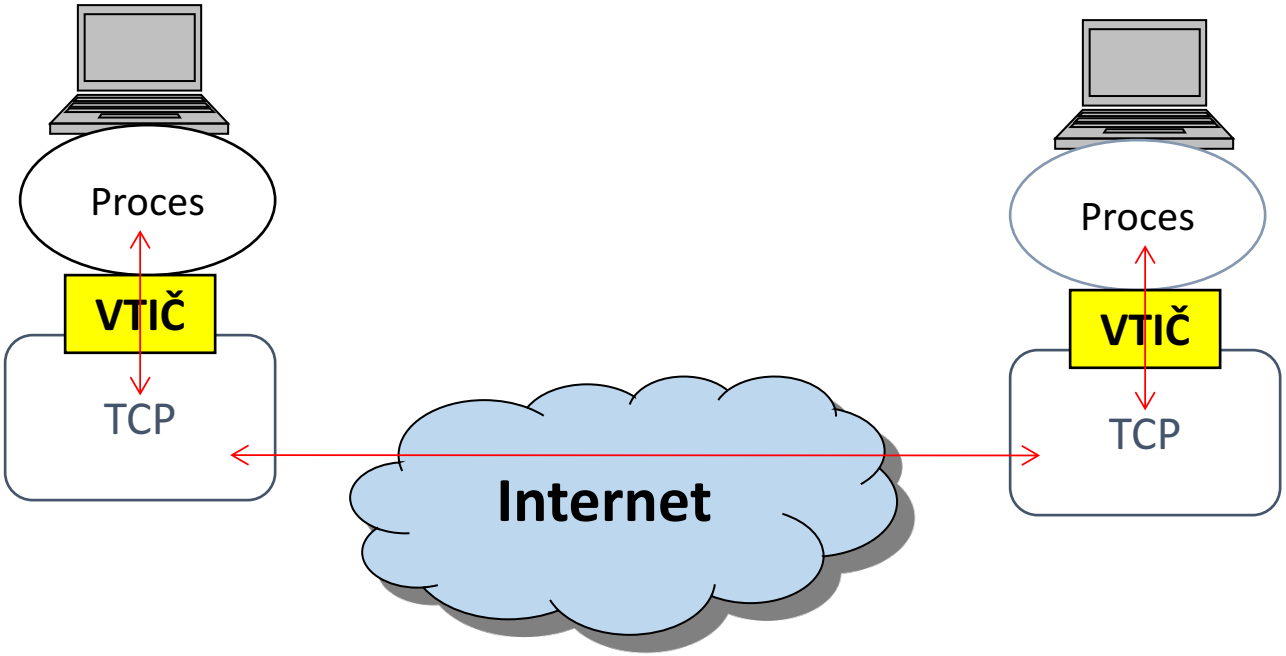
Storitve transportne plasti

- **razlika** med omrežno in transportno plastjo
  - omrežna plast: logična povezava med *končnimi sistemi*
  - transportna plast: logična povezava med *procesi*

- **storitve** transportne plasti:
  1. so **omejene** s storitvami nižje (t. j. omrežne) plasti.
     - *analogija - kako sta Ana in Eva omejeni?*
  2. vsak transportni protokol lahko **zagotavlja svojo množico** storitev
     analogija: kaj, če namesto Ane in Eve dostavljata Katja in Luka?
     - TCP: zanesljiva, povezavna storitev, ima nadzor zamašitev
     - UDP: best-effort (nezanesljiva), nepovezavna storitev
  3. v Internetu nimamo naslednjih storitev: zagotovljen čas dostave, zagotovljena pasovna širina

Kako komunicirati s procesom (aplikacijo)?

- vsak proces (~ aplikacija) 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**



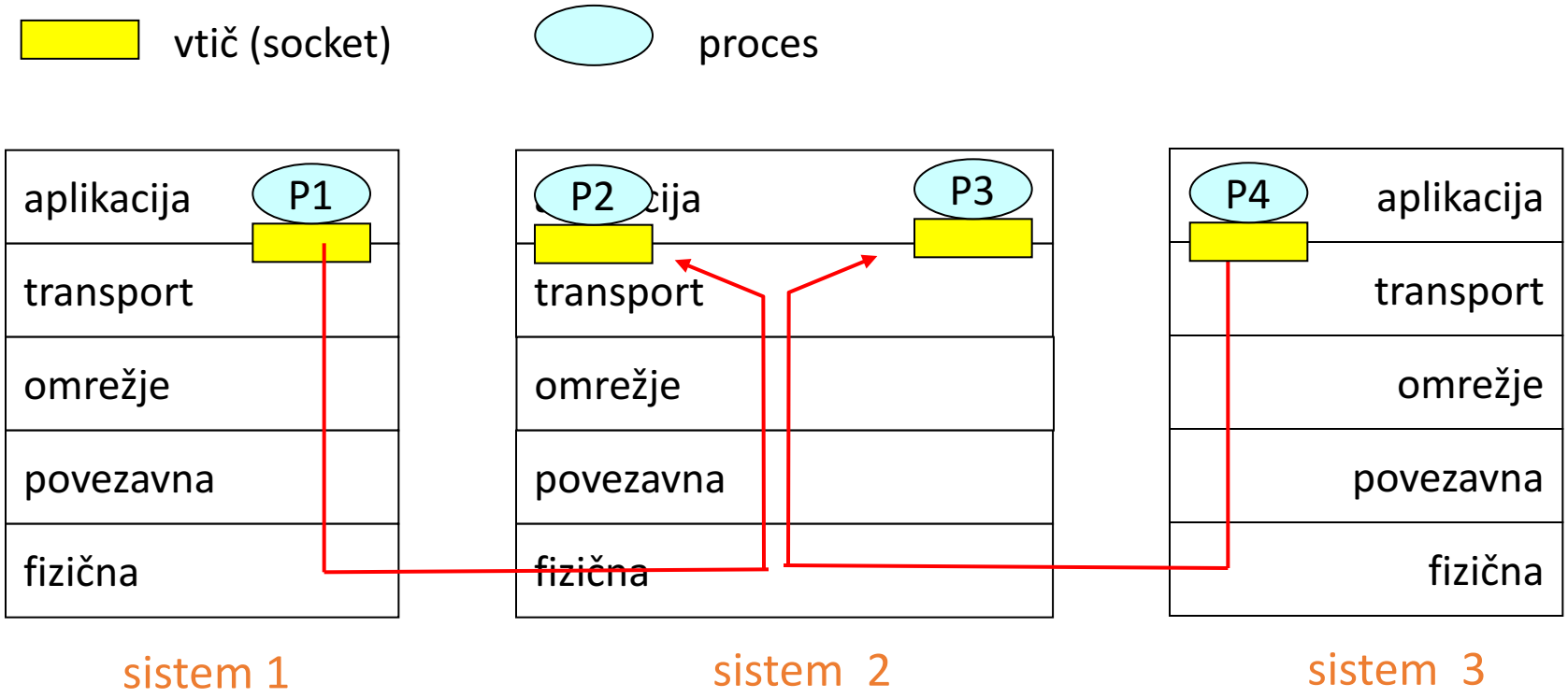
Kako poteka demultipleksiranje?

- vsak transportni segment potuje znotraj svojega paketa IP
- **transportni segment v glavi 16-bitni polji:** številki vrat *izvora* (oznaka procesa, ki pošilja) in *ponora* (oznaka procesa na ciljni strani)

oblika segmenta TCP/ datagrama UDP

Multipleksiranje in demultipleksiranje

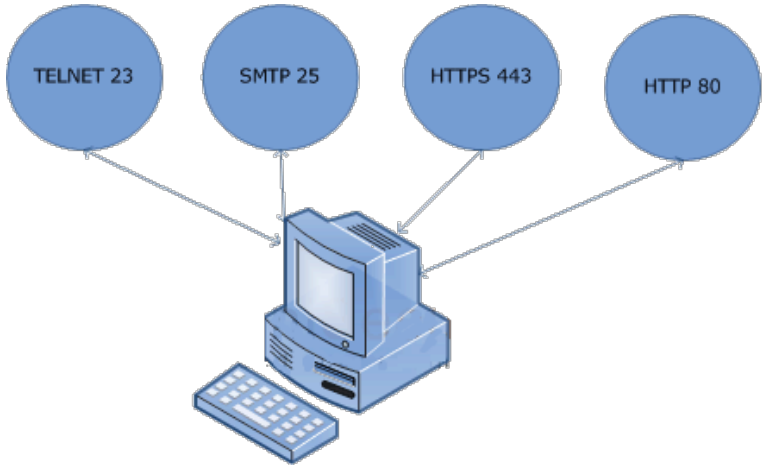
- **pošiljatelj**: pobira podatke z več vtičev (*socket*), opremi jih z glavo, pošlje
- **prejemnik**: segmente razdeli ustreznim vtičem



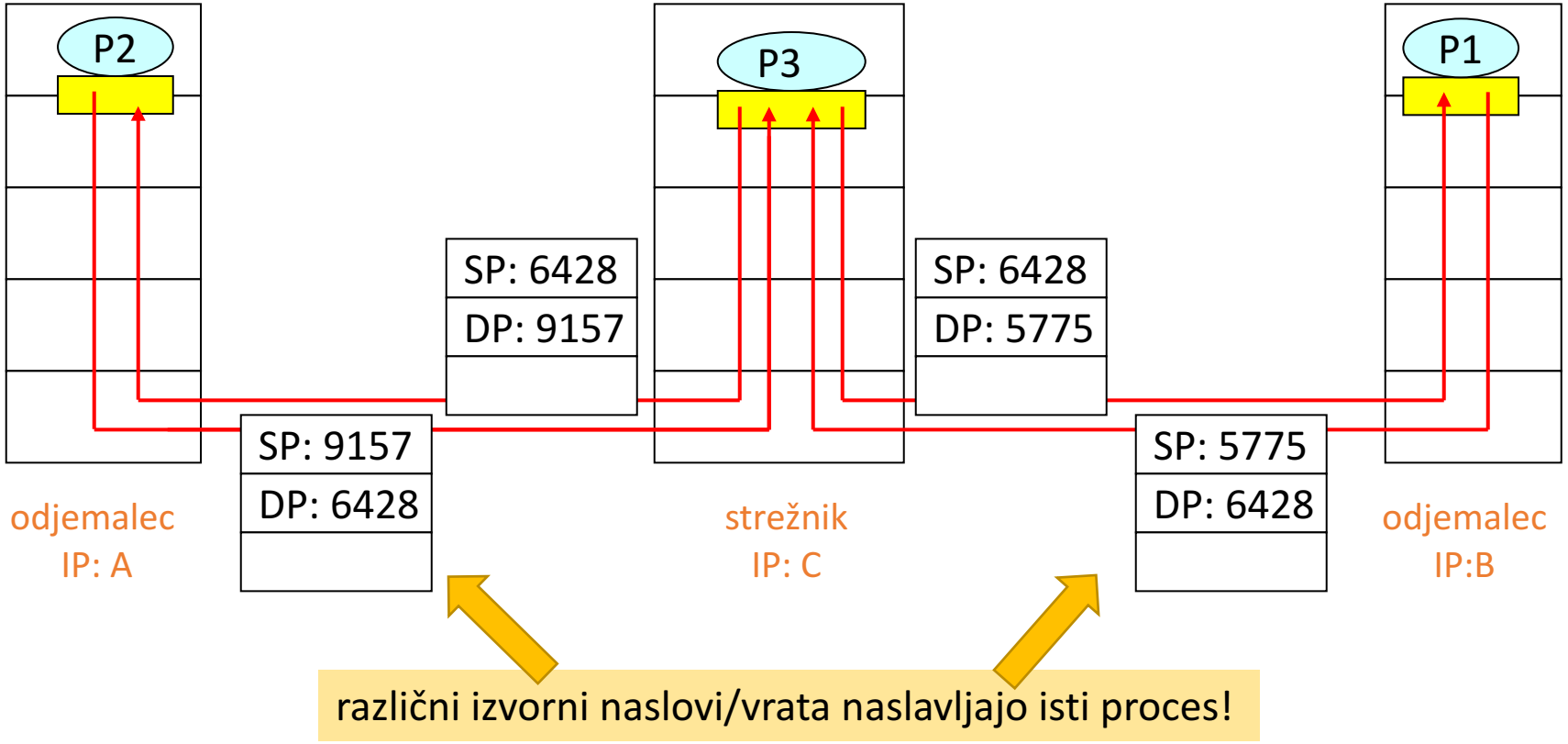
Kako nasloviti proces na drugi strani?

- **za enolično 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
  - več: www.iana.org



Nepovezavno demultipleksiranje (UDP)



SP = **s**ource **p**ort, izvorna vrata (predstavlja naslov za odgovor)
DP = **d**estination **p**ort, ciljna vrata (za naslavljanje vtiča ciljnega procesa)

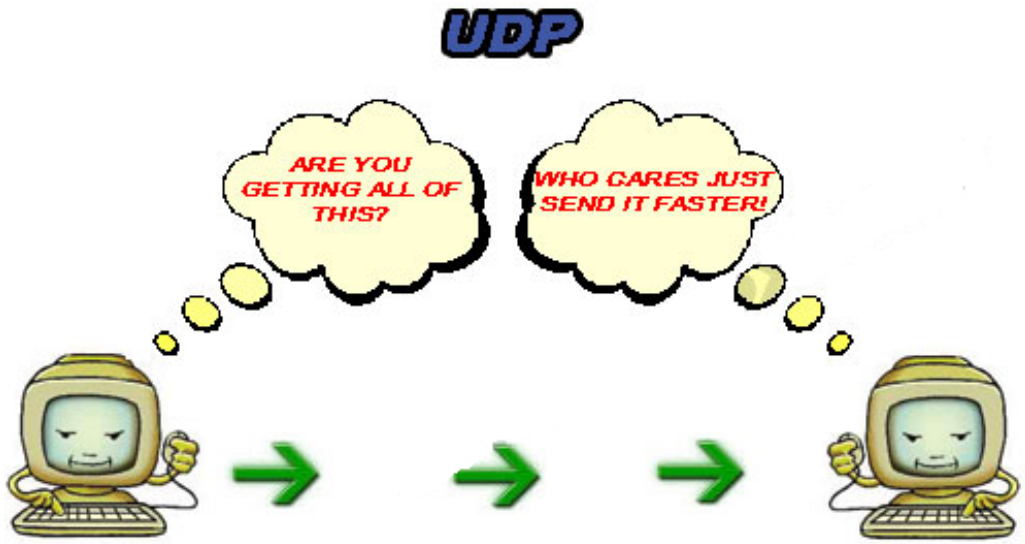
Varnost: Napad portscan

*Napad **portscan** (pregled vrat) je namenjen ugotavljanju, na katerih vratih se strežnik odziva.*

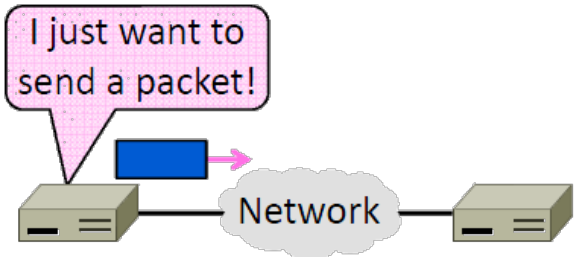
Nudi vpogled v procese, ki tečejo na strežniku.

S poznavanjem šibkih točk strežniške programske opreme (npr. OS, SQL server...) lahko napadalec ogrozi delovanje sistema.

Nepovezavni transport: UDP (*User Datagram Protocol*)



Storitve protokola UDP



LASTNOSTI

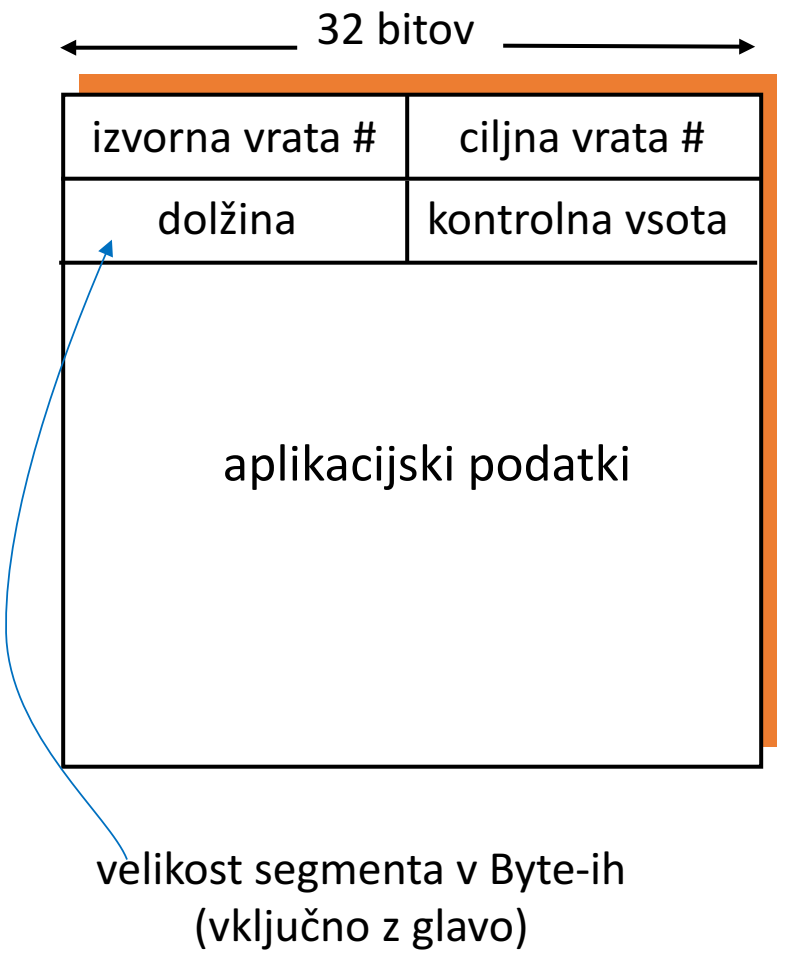
- nudi le "best-effort" storitev:
  - izgubljeni datagrami ("segmenti")
  - ne zagotavlja vrstnega reda
- nepovezaven (nima rokovanja)
- nima nadzora zamašitev

PREDNOSTI

- okleščen, najbolj osnoven prenosni protokol brez dodatkov
- hiter, učinkovit, lahek, minimalističen (ne hrani stanja o povezavi, medpomnilnikov, ni rokovanja)
- majhna glava datagrama (samo 8B), torej manj režije

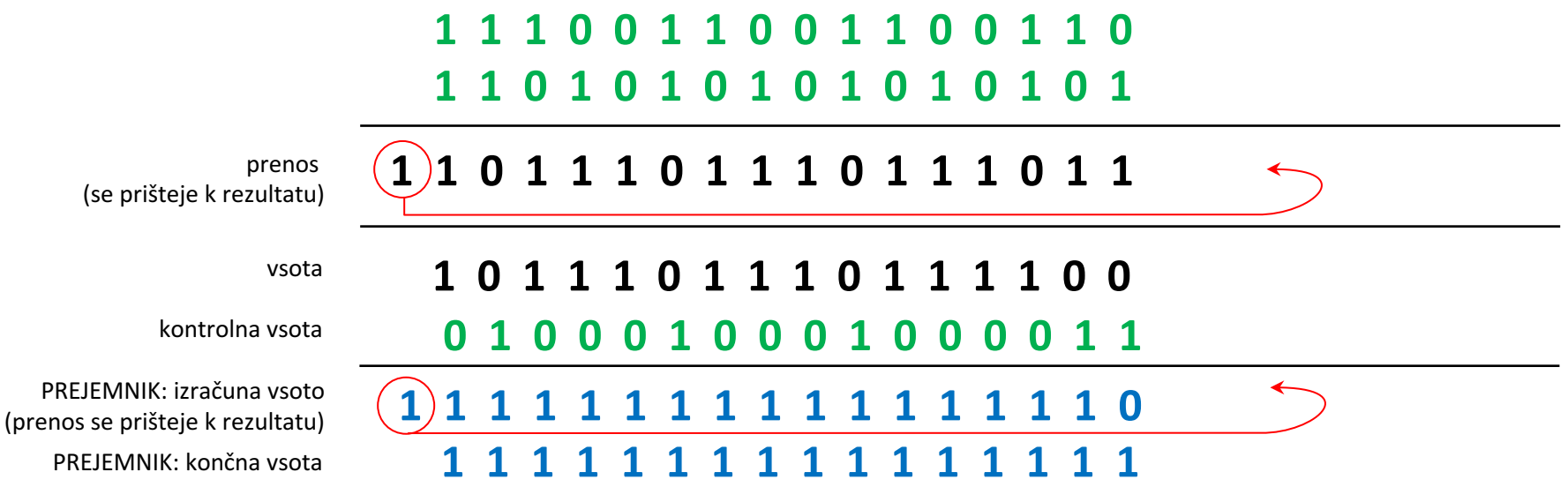
UDP datagram

- 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

- 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

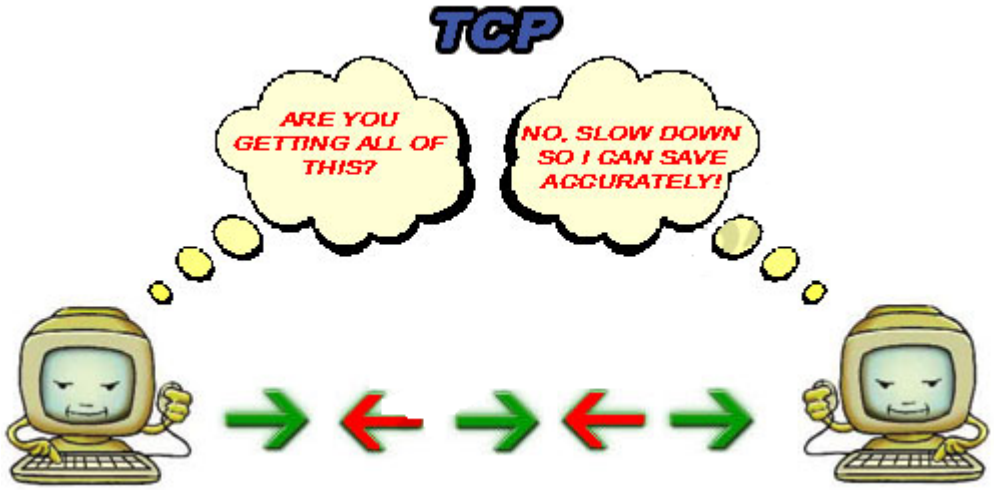


UDP: internetna kontrolna vsota

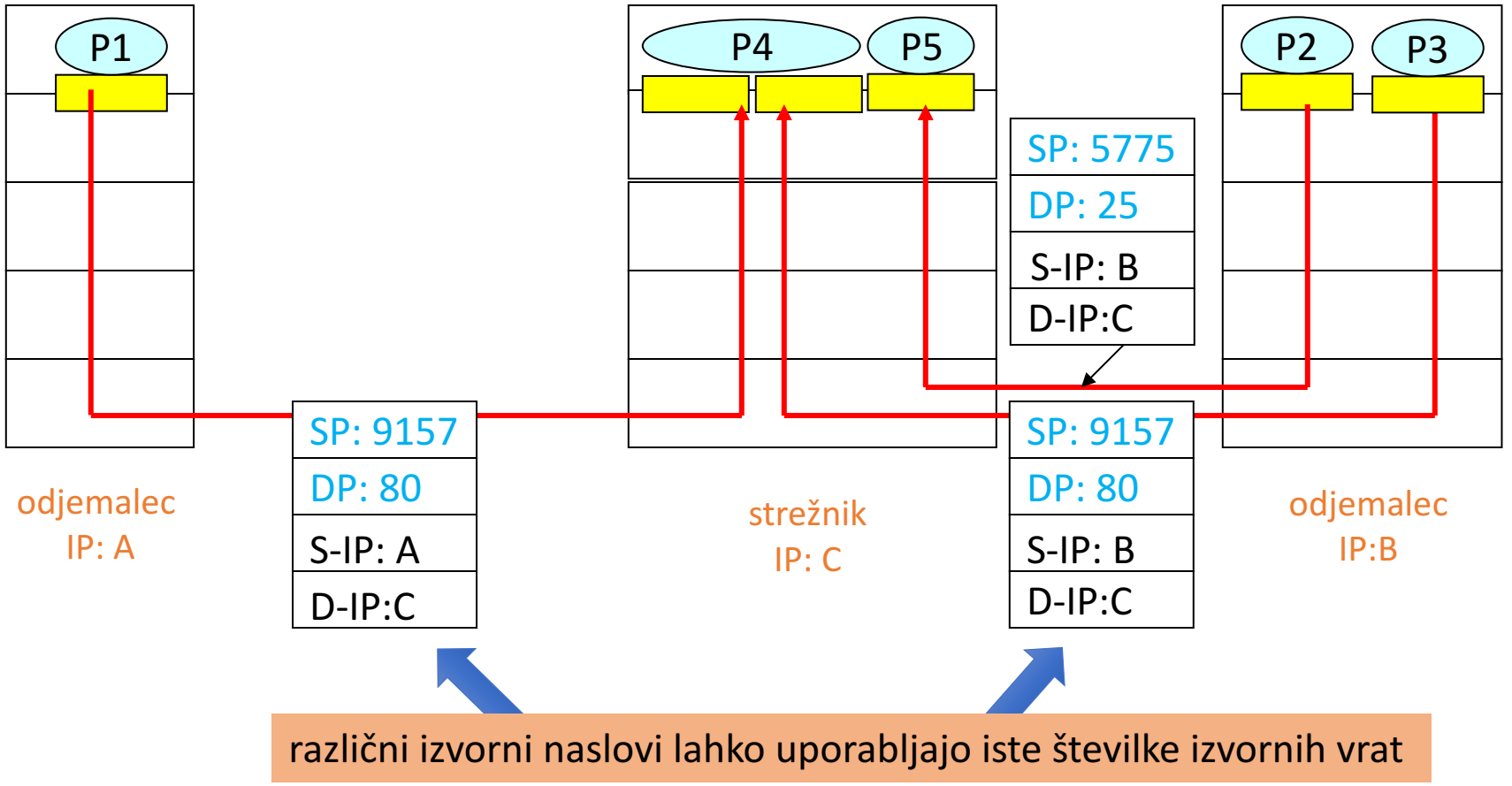
- zakaj datagram vsebuje kontrolno vsoto?
  - ni zagotovila, da nižji protokol na posameznih povezavah zagotavlja zaznavanje in odpravljanje napak
  - do napak lahko pride tudi pri hranjenju segmenta v spominu usmerjevalnika in ne nujno pri prenosu (prenosni protokol zagotavlja samo zaznavanje napak pri prenosu)
  - UDP kontrolna vsota je namenjena preverjanju pravilnosti med izvornim in ciljnim procesom, ne pa pri potovanju po posameznih povezavah (t. i. princip končnih sistemov, *end-to-end argument/principle*)

    - Vprašanje: kje smo dosedaj že omenili princip končnih sistemov?

Povezavni transport: TCP (*Transfer Control Protocol*)



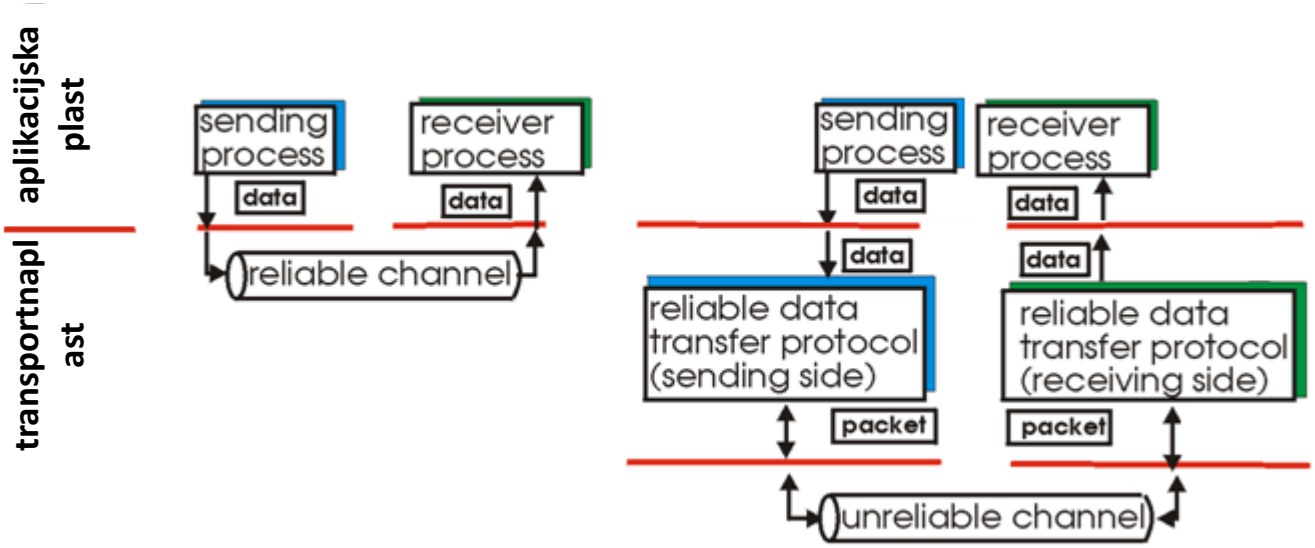
Povezavno demultipleksiranje (TCP)



SP = **s**ource **p**ort, izvorna vrata (predstavlja naslov za odgovor)
DP = **d**estination **p**ort, ciljna vrata (za naslavljanje vtiča ciljnega procesa)

Protokol TCP

- Potrebujemo protokol, ki zagotavlja zanesljivo dostavo z uporabo nezanesljivega kanala (!). Kaj mora tak protokol nuditi?
  - podatki se ne okvarijo (zaznavamo zamenjave bitov 0 <-> 1)
  - podatki se ne izgubljajo (zaznavamo izgube, ponovno pošiljamo)
  - podatki so dostavljeni v pravilnem zaporedju (urejanje)



koncept zanesljive storitve

dejanska implementacija storitve

Zagotavljanje zanesljivosti: potrjevanje

- Paket se okvari
- Paket se izgubi – ni potrditve
- Izguba potrditve (pride podvojen paket)
- Prekratek interval časovne kontrole (duplikat pride)
- Posredno potrjevanje – samo ACK (optimizacija)
- Neučinkovitost sprotnega potrjevanja: tekoče pošiljanje
  - Ponavljanje zaporedja
  - Selektivno ponavljanje

- Kontrolna vsota, potrditve ACK-NACK
- Časovna kontrola (ponovitev)
- Številčenje paketov omogoča zaznavanje duplikatov
- Številčenje paketov, številčenje potrditev

- Tekoče pošiljanje brez čakanja na sprotne potrditve (drseče okno)

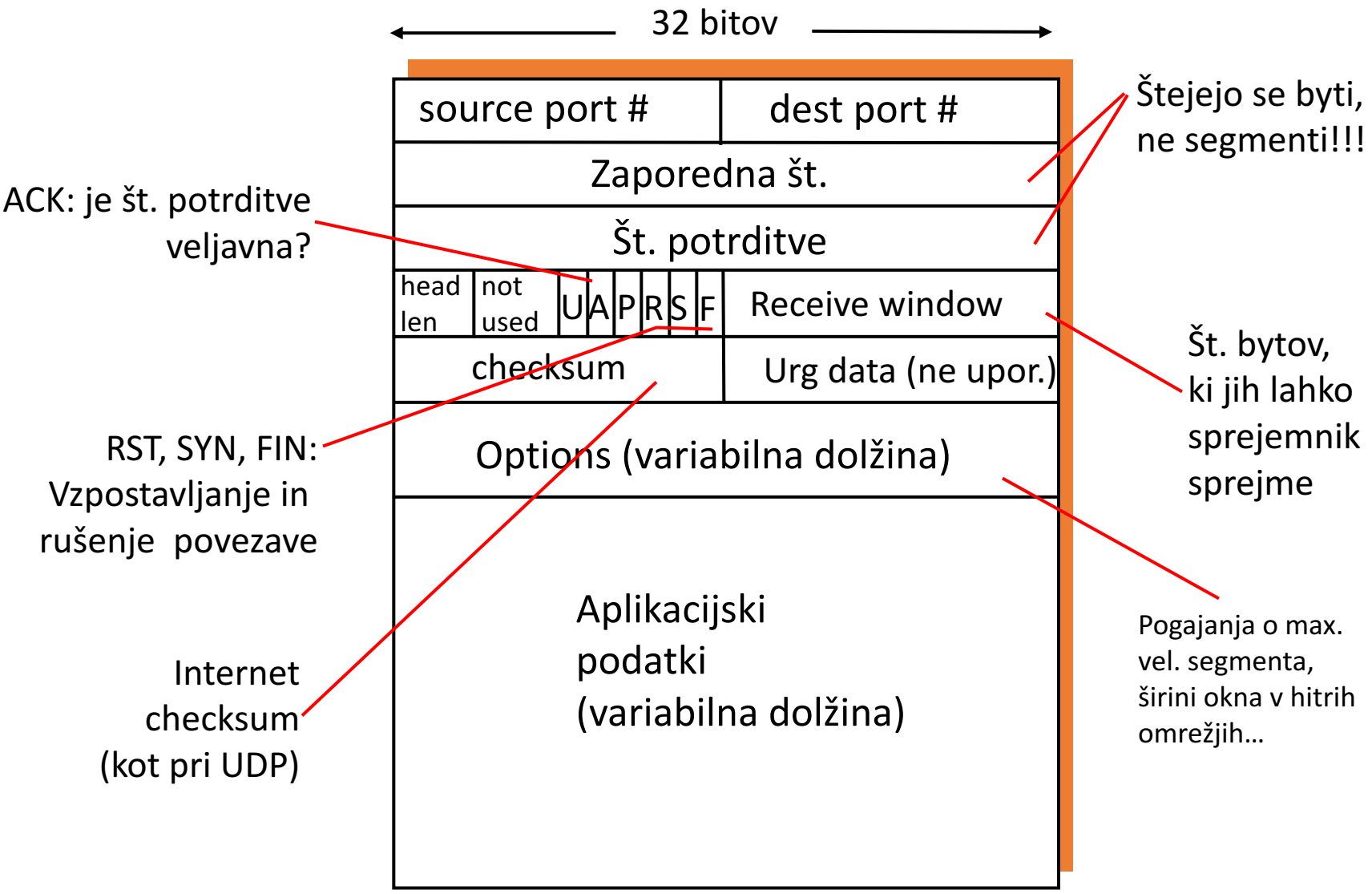
Lastnosti protokola TCP

- izvaja se med dvema točkama (point-to-point):
  - **en pošiljatelj, en sprejemnik**
- znotraj TCP povezave izvaja **dvosmerni promet**
  - (full duplex, omejitev MSS)
- nudi **zanesljiv**, urejen prenos podatkov
- je **povezavni protokol**
  - vzpostavitev/rušenje zveze
- ima **kontrolo pretoka** (angl. *flow control):*
  - pošiljatelj ne preobremeni prejemnika
- ima **kontrolo zamašitev** (angl. *congestion control):*
  - pošiljatelj ne preobremeni omrežja
- uporablja **tekoče pošiljanje**:
  - velikost okna se avtomatsko določa glede na kontrolo pretoka in kontrolo zamašitve

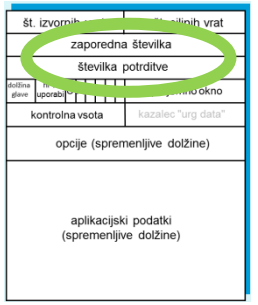
Simulacija potrjevanja

http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/

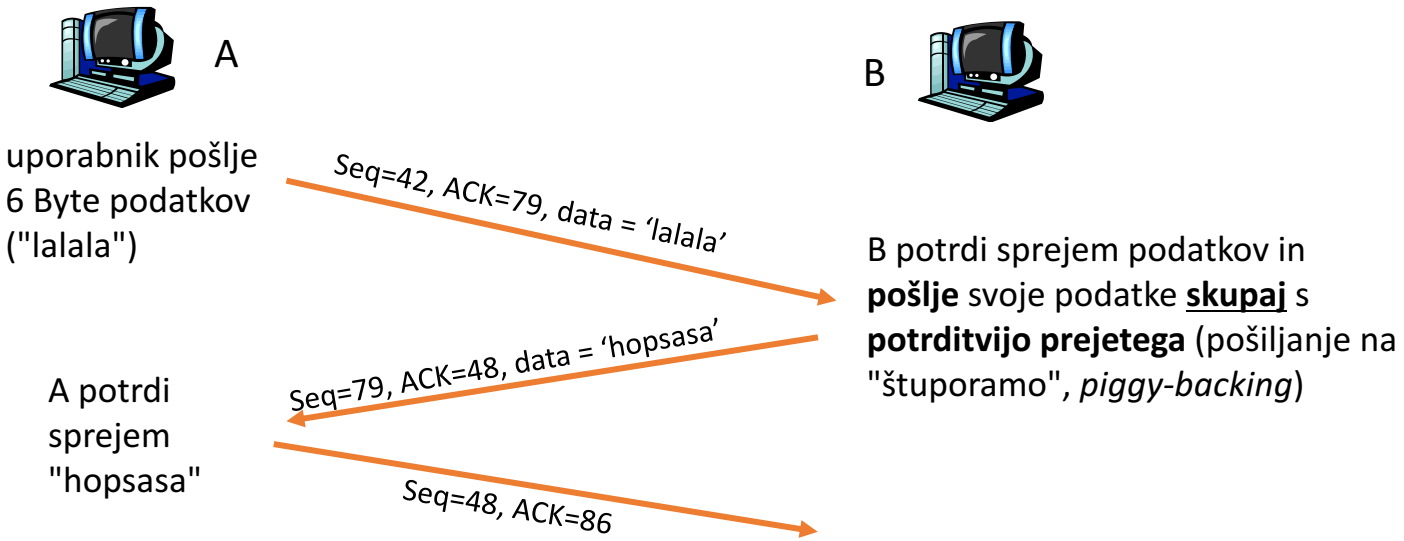
TCP segment:

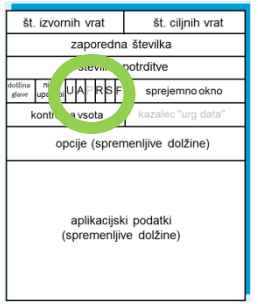

32 bitov
source port #
dest port #
Zaporedna št.
Št. potrditve
head len
not used
U
A
P
R
S
F
Receive window
checksum
Urg data (ne upor.)
Options (variabilna dolžina)
Aplikacijski podatki (variabilna dolžina)
Štejejo se byti, ne segmenti!!!
ACK: je št. potrditve veljavna?
RST, SYN, FIN: Vzpostavljanje in rušenje povezave
Št. bytov, ki jih lahko sprejemnik sprejme
Internet checksum (kot pri UDP)
Pogajanja o max. vel. segmenta, širini okna v hitrih omrežjih…

Številčenje segmentov in potrditev



- pošiljatelj in prejemnik najprej VZPOSTAVITA ZVEZO. Povezava je nato **dvosmerna** (vsak lahko pošilja drugemu podatke in potrditve)
- pošiljatelj lahko v enem segmentu **istočasno** pošlje **nove podatke in potrditev** (ACK) prejšnjega segmenta
- **številke pomenijo:**
  - **SEQ (zaporedna številka):** zap**.** številka prvega byte-a v segmentu
  - **ACK (potrditev):** zap**.** številka naslednjega pričakovanega byte-a



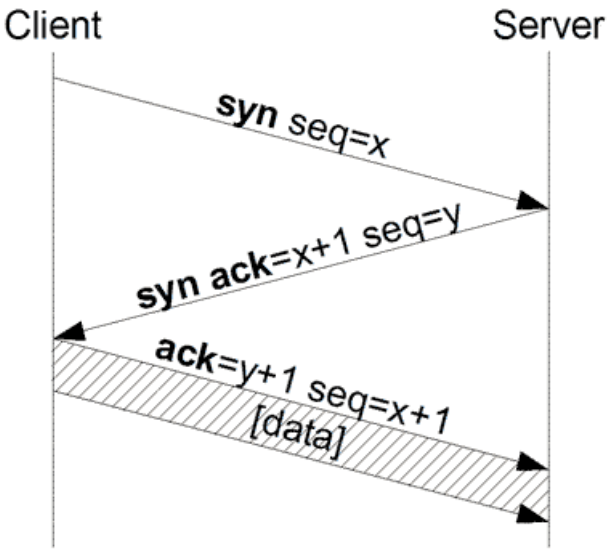


TCP: vzpostavljanje povezave

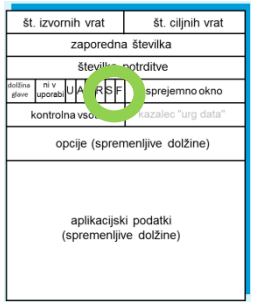
- pošiljatelj in prejemnik pred pošiljanjem izvedeta rokovanje (handshake), v katerem izmenjata parametre:
  - začetne pričakovane zaporedne številke (naključno določene)
  - velikosti medpomnilnikov (za kontrolo pretoka)

- trojno rokovanje (*three-way handshake*)

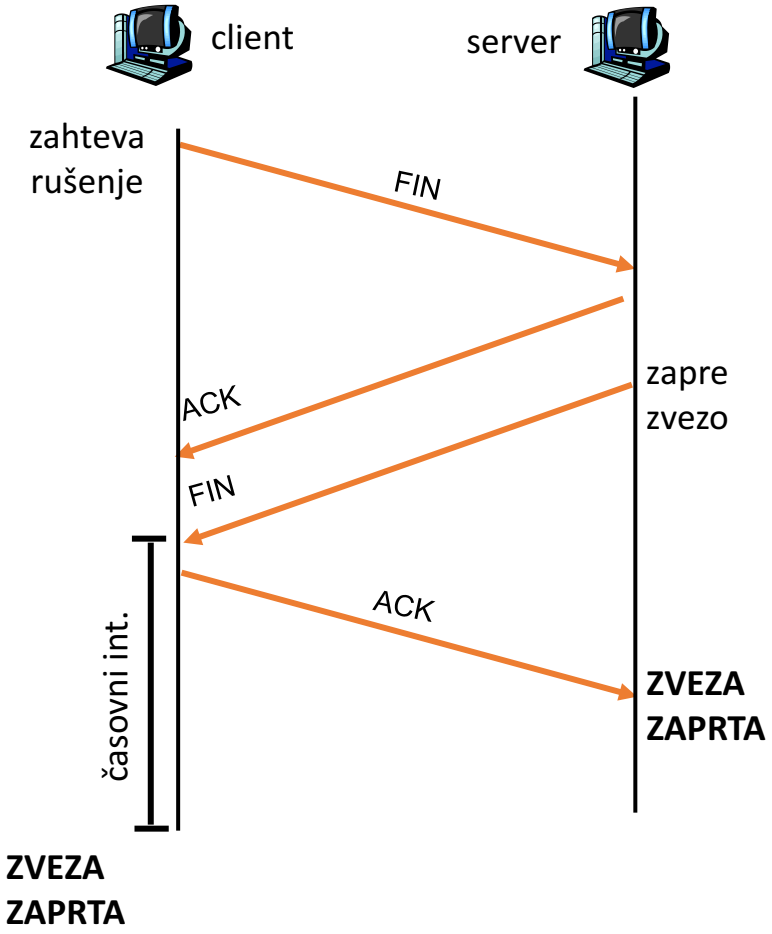
1. Odjemalec pošlje segment z zastavico **SYN** (sporoči začetno številko segmenta, ni podatkov)
2. Strežnik vrne segment **SYN ACK** (rezervira medpomnilnik, odgovori z začetno številko svojega segmenta)
3. Odjemalec vrne **ACK**, lahko že s podatki (potrjevanje "štuporamo")



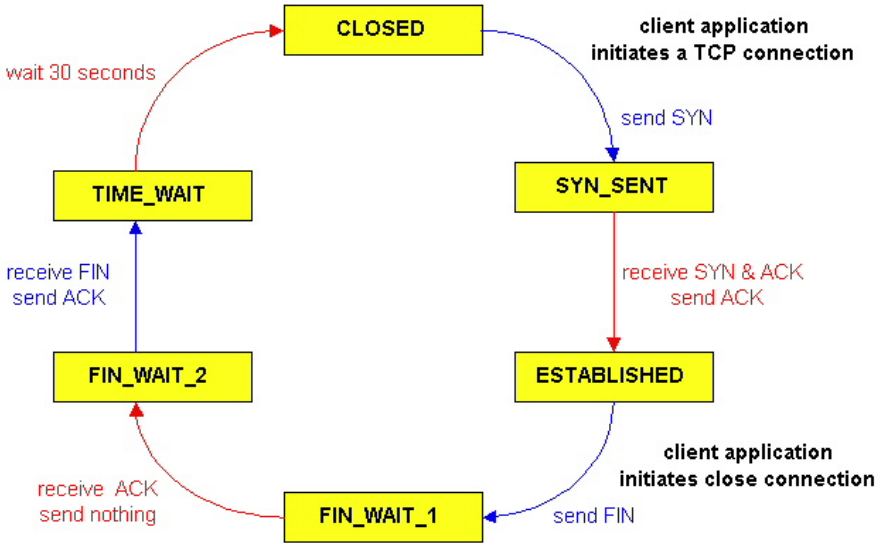
TCP: rušenje povezave



1. odjemalec pošlje segment TCP FIN strežniku (zastavica!)
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

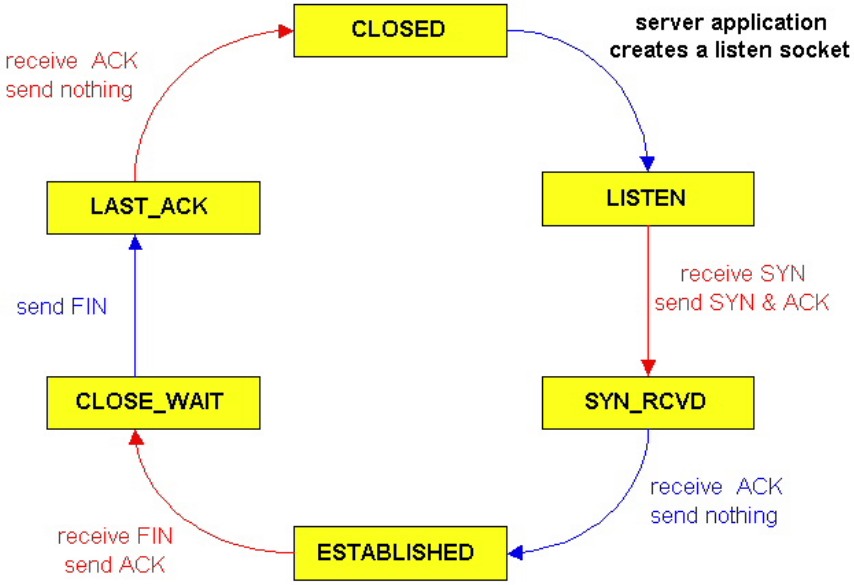


Življenjska cikla odjemalca in strežnika



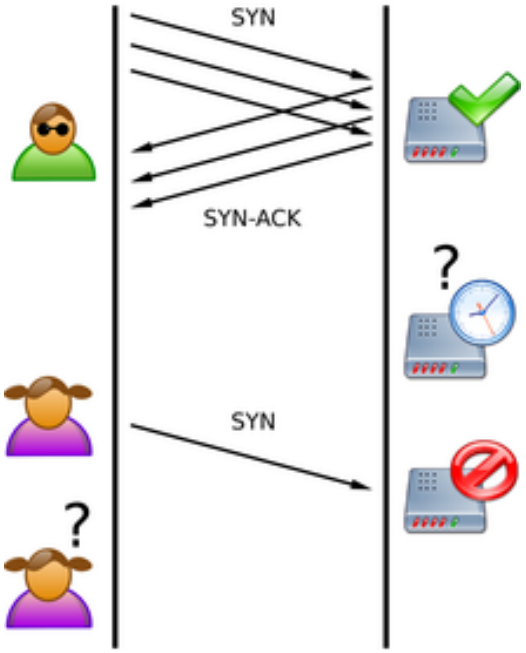
TCP odjemalec

TCP strežnik



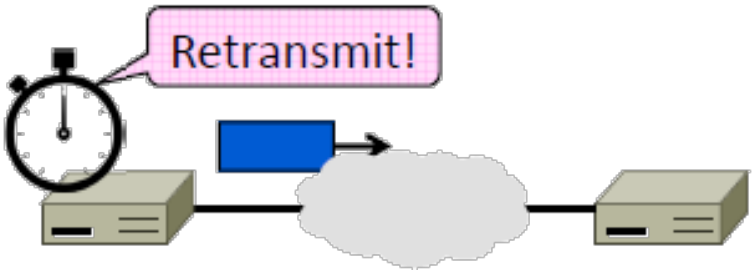
Varnost: napad SYN FLOOD

- Napadalec pošlje strežniku veliko število paketov (TCP SYN). Strežnik vsakič rezervira del svojega medpomnilnika
- Napadalec ne zaključi rokovanja z ACK., prostor na strežniku ostane zaseden do timeouta.
- Zaradi velikega števila odprtih povezav strežniku zmanjka prostora in ne more več sprejemati novih povezav - DoS (angl. *denial of service*)
    - porazdeljeni DoS napad (DDoS): pošiljanje TCP SYN iz več virov



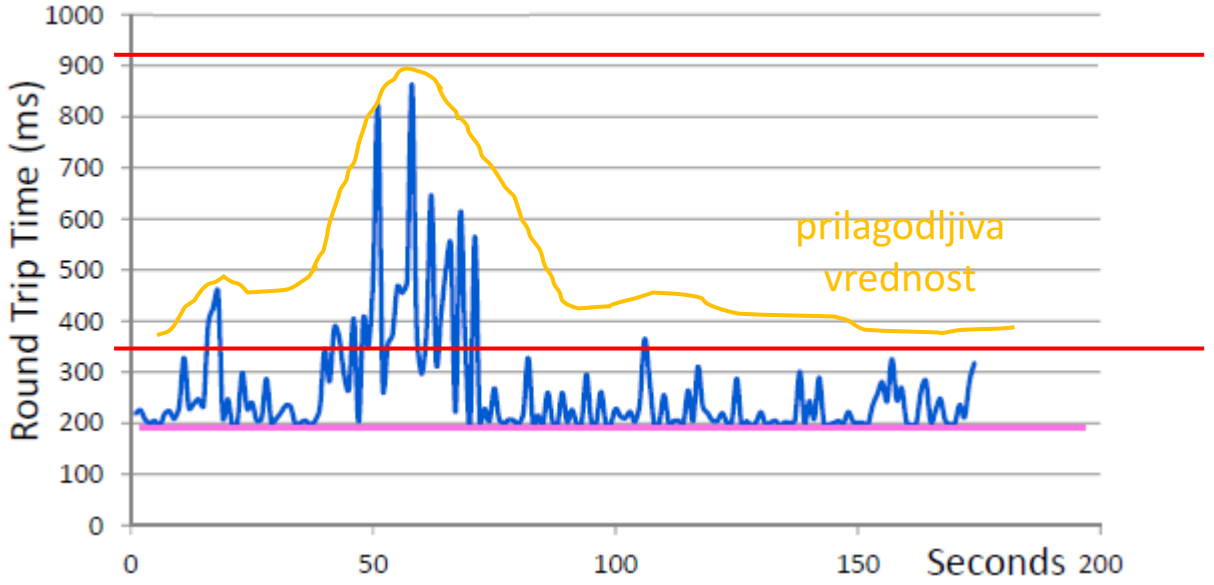
Nastavitev časovne kontrole v TCP

- **štoparica (časovna kontrola)**: potrebna za zanesljivo dostavo - če se izgubi paket ali potrditev, sproži ponovno pošiljanje
- Kakšna je primerna dolžina čakalnega intervala?
  - **daljši od časa vrnitve** (RTT, *Round Trip Time*) = čas za pot paketa od pošiljatelja do prejemnika in potrditve nazaj
    - če je prekratek, imamo preveč ponovnih pošiljanj
    - če je predolg, prepočasi reagiramo na izgubljene segmente

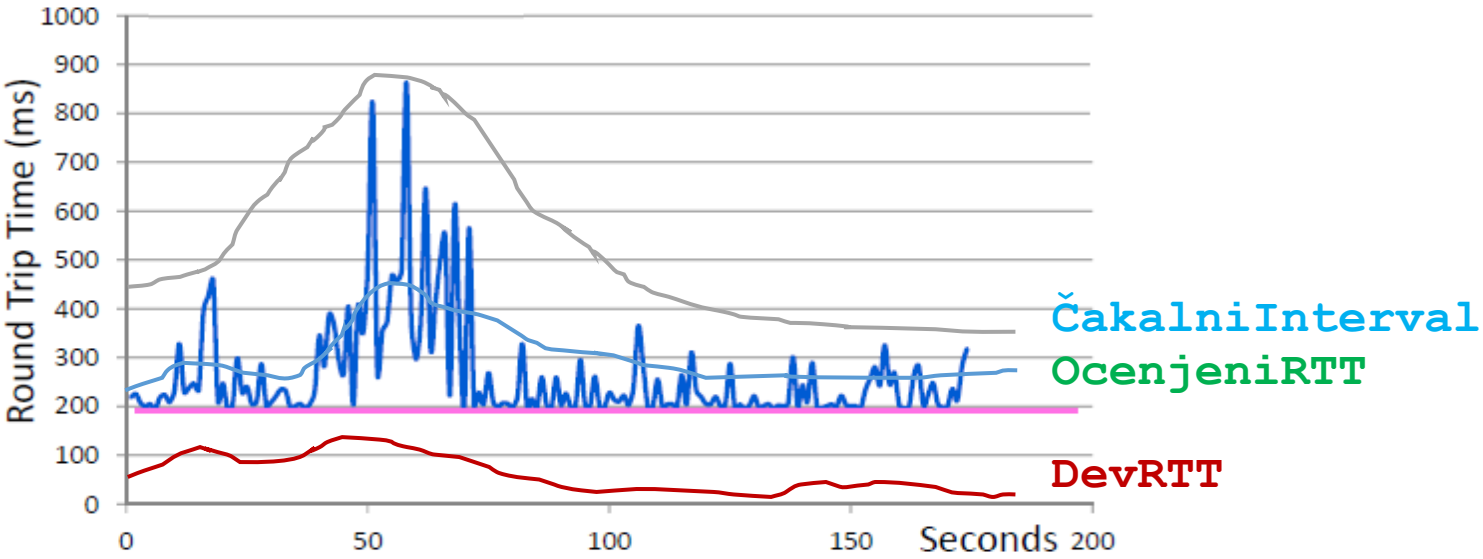


Primer ocenjevanja RTT

- 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 in čakalnega int.



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

- Kakšne vrste ponavljanje uporablja TCP? Ali
  - ponavljanje zaporedja (N nepotrjenih - *go-back-N*) ali
  - potrjevanje posameznih (*selective repeat*)?

- ODGOVOR: **kombinirano rešitev** obeh
  - podoben ponavljanju N nepotrjenih (kjer je štoparica le 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

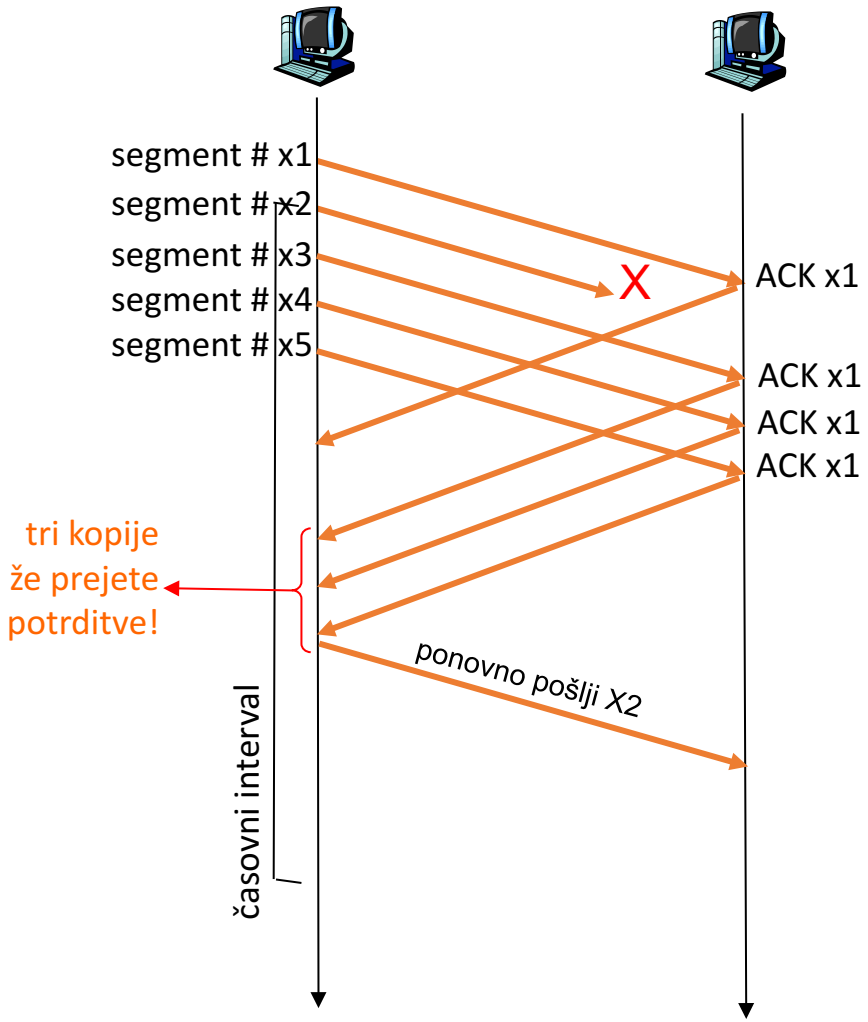
Posebnosti pri potrjevanju

Dogodek pri prejemniku	Odziv prejemnika
Sprejem segmenta s pričakovano številko, vsi prejšnji že potrjeni.	Počakaj na nasl. segment še max 500ms. • Če pride, oddaj **zakasnjeno potrditev obeh** (delayed ACK). • Če ne pride, potrdi samo prejetega.
Isto kot zgoraj, a prejšnji segment še ni nepotrjen.	Takoj pošlji **kumulativno potrditev obeh**.
Sprejem segmenta s previsoko številko (**zaznamo vrzel**)	Takoj potrdi zadnji v zaporedju sprejeti segment (pošlji duplikat ACK).
Sprejem segmenta z najnižjo številko iz vrzeli (**polnjenje vrzeli**)	Takoj potrdi segment.

Hitro ponovno pošiljanje (*fast retransmit*)

- običajno se 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



TCP: algoritem za zanesljiv prenos

osnova = začetna zaporedna številka;

naslednji = začetna zaporedna številka;

while(true) {

Če: aplikacija pošlje podatke

- naredi TCP segment s številko *naslednji*, sproži štoparico
- segment predaj protokolu IP
- *naslednji* <- *naslednji* + dolžina podatkov

Če: poteče časovni interval

- ponovno oddaj nepotrjen segment z najmanjšo zaporedno številko
- sproži štoparico

Če: prejme potrditev s številko y

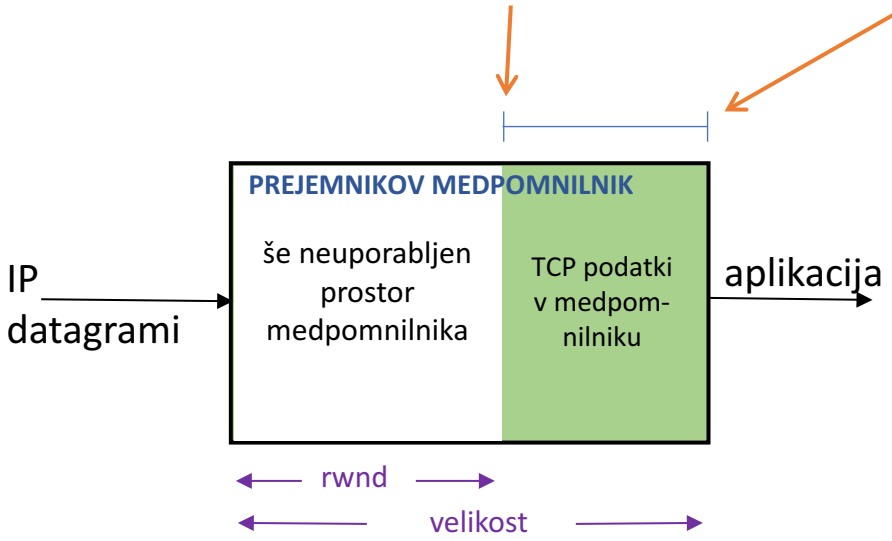
- če y > *osnova*: /* potrditev zaporedja do vključno y */
  osnova = y; /* prestavi okno naprej */
  sproži štoparico za naslednji še nepotrjeni paket
- sicer /* prejeli smo duplikat že prejete potrditve */
  poveča števec duplikatov ACK za y;
  če je števec duplikatov = 3, ponovno pošlji segment y
  ponastavi štoparico za segment y /* to je TCP fast retransmit*/

}

Kontrola pretoka TCP

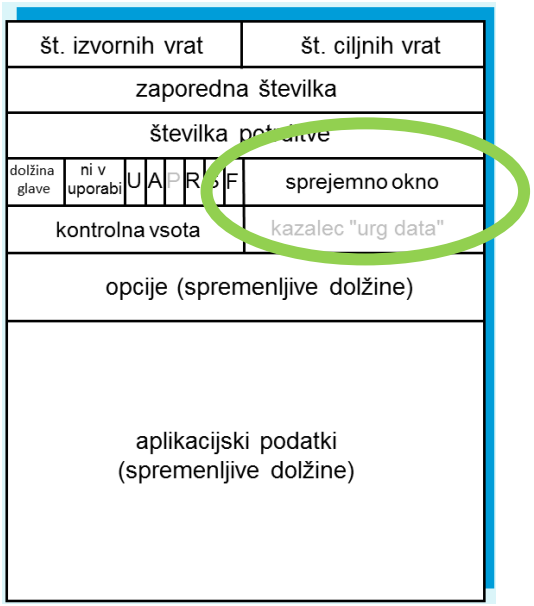
- usklajevanje med pošiljateljem in prejemnikom:
  - pošiljatelj ne sme pošiljati hitreje, kot lahko prejemnik bere,
  - da ne povzroči prekoračitve medpomnilnika (prejemnikov prostor, kjer se začasno shranjujejo prejeti segmenti pred predajo aplikaciji)
- razpoložljiv prostor medpomnilnika, **sprejemno okno** – receive window:

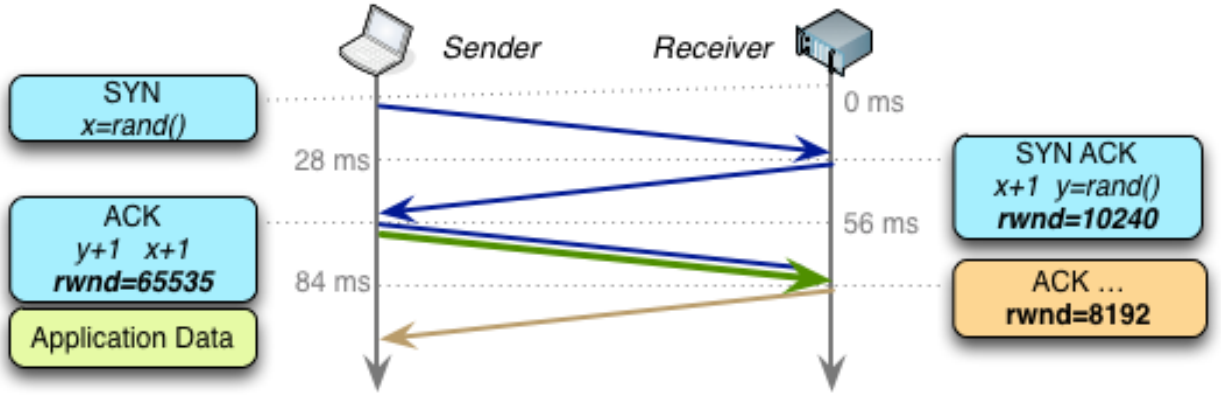
```
rwnd = velikost - [LastByteRcvd - LastByteRead]
```



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

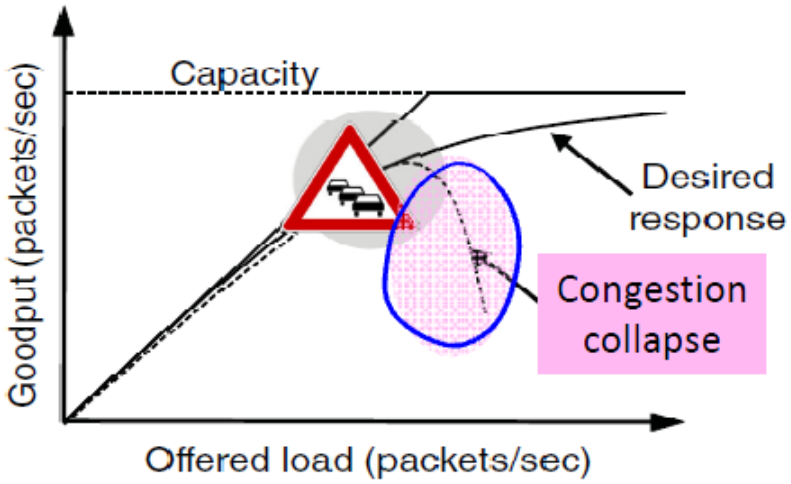




Nadzor zamašitev

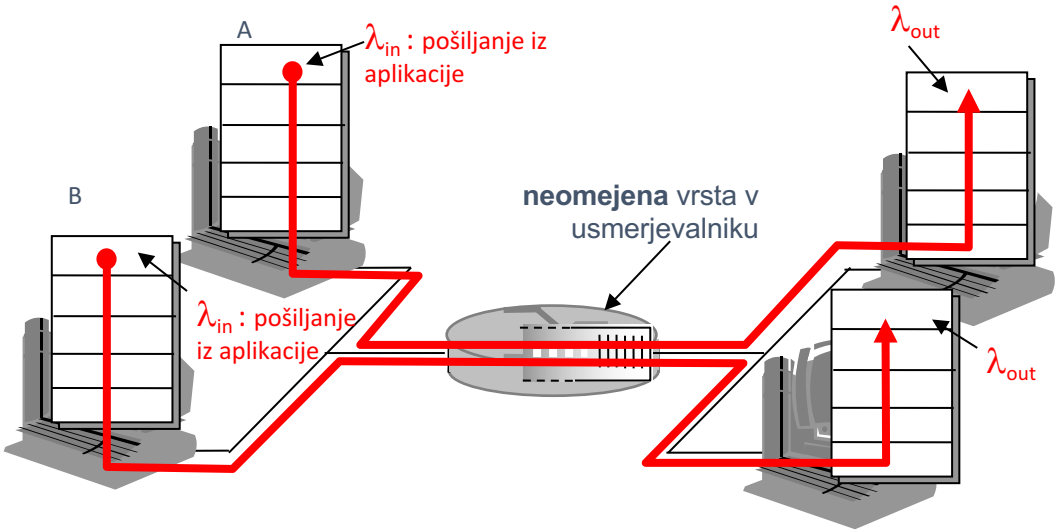
- zamašitev: stanje preobremenjenosti omrežja, ko več virov naenkrat prehitro pošilja preveč podatkov za dano omrežje
- posledica zamašitve:
  - izguba segmentov (prekoračitve medpomnilnika v usmerjevalnikih)
  - velike zakasnitve (čakalne vrste v usmerjevalnikih)

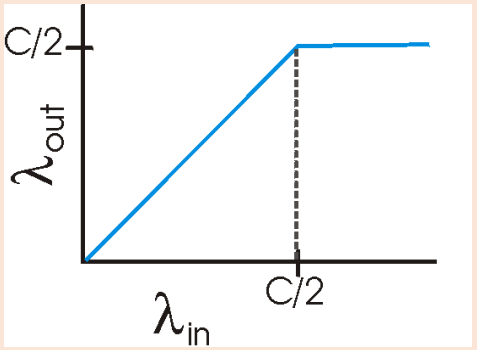
- ni isto kot nadzor pretoka!



Zamašitev – primer 1

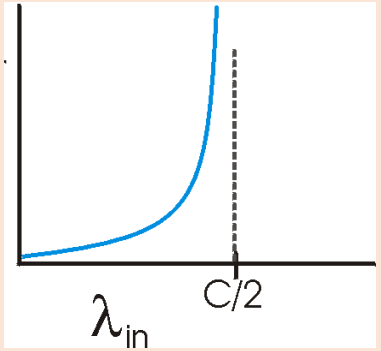
- dva pošiljatelja, neomejen pomnilnik v usmerjevalniku (za čakalno vrsto)
- C - kapaciteta kanala





pretok:

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

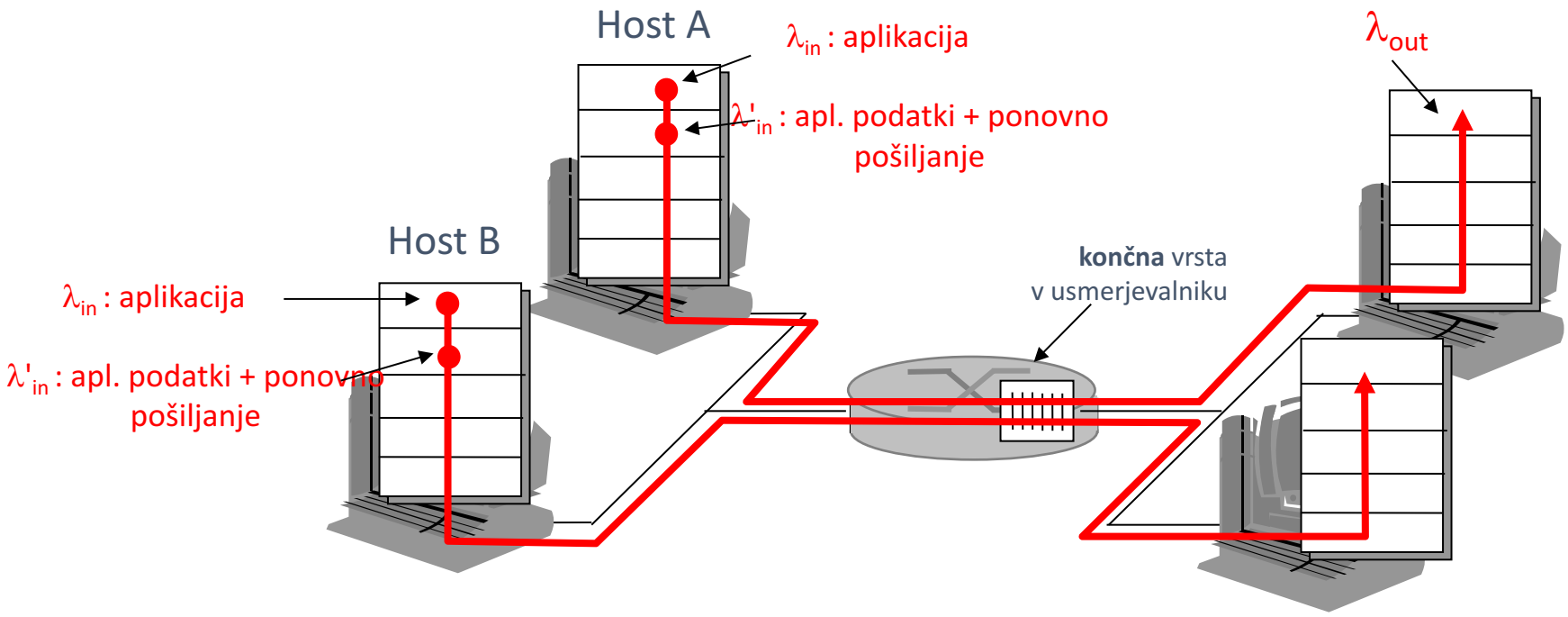


zakasnitev:

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

Zamašitev – primer 2

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



Zamašitev – primer 2

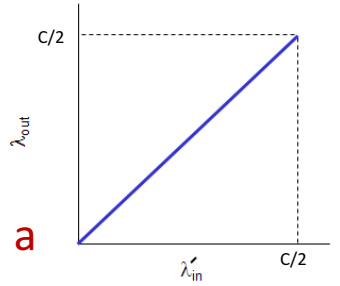
Preučimo tri scenarije:

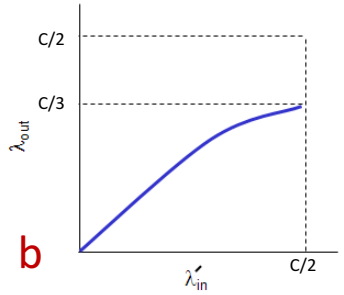
a. segment oddamo le, ko je prostor v vrsti, tako da ni izgub (v praksi to ni možno, ker tega ne vemo)

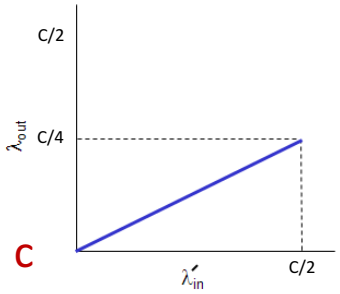
b. dogajajo se izgube paketov in ponovna pošiljanja

c. ponovna pošiljanja tudi zaradi velikih zakasnitev

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

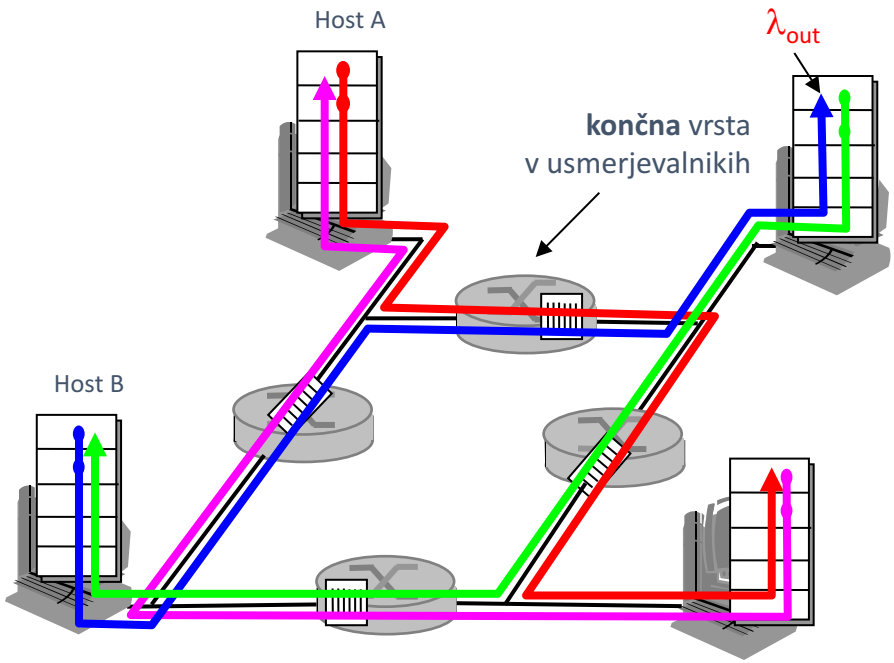


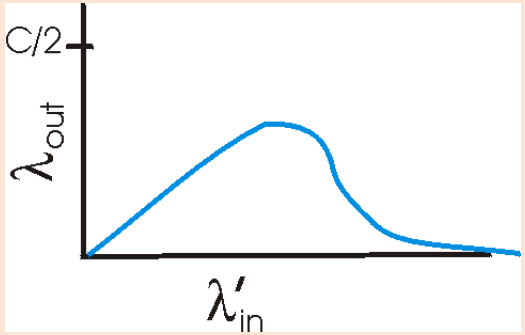




Zamašitev – primer 3

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



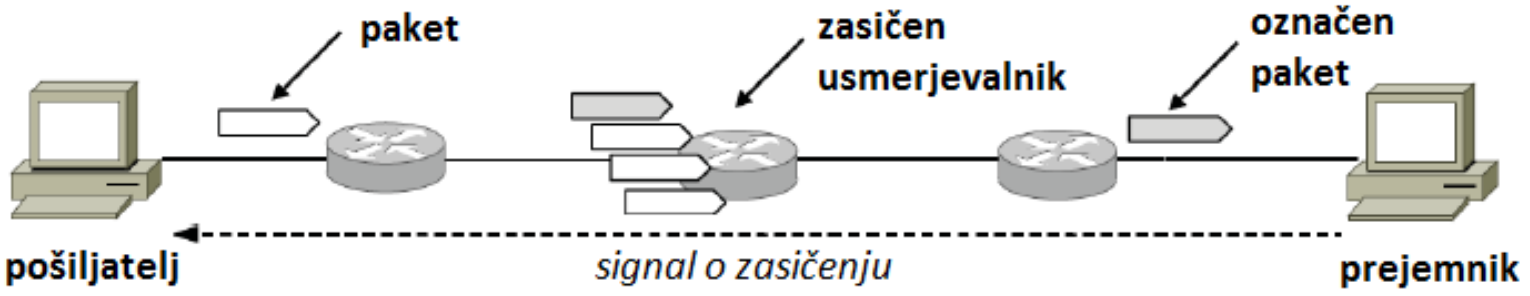


pošiljanje z večjo hitrostjo polni čakalno vrsto v neskončnost

Kako nadzorovati zamašitve?

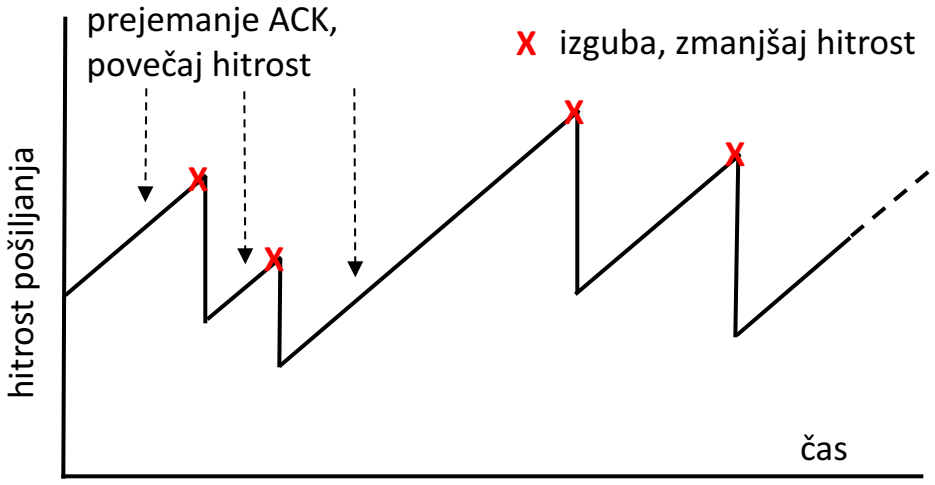
dva pristopa:

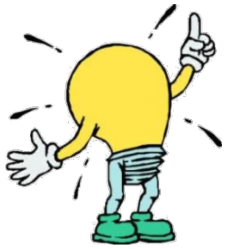
1. na podlagi **končnih sistemov** (end-to-end)- **to tehniko uporablja TCP**.
   - bodisi prejemnik sporoči pošiljatelju, da so usmerjevalniki na poti sporočili zamašitev
   - bodisi pošiljatelj opazuje čas do prejema potrditve
2. z uporabo **omrežnih storitev**: usmerjevalniki v omrežju obvestijo pošiljatelja, da je prišlo do zamašitve
   - uporaba obvestila o zamašitvi (ECN – explicit congestion notification) v IP/TCP
   - usmerjevalnik nastavi ustrezen bit in sporoči sprejemljivo hitrost oddajanja (npr. pri ATM)



TCP nadzor zamašitev

- **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 zamašitve, poveča hitrost
  - če se segment izgubi, je to posledica zamašitve, zmanjšaj hitrost



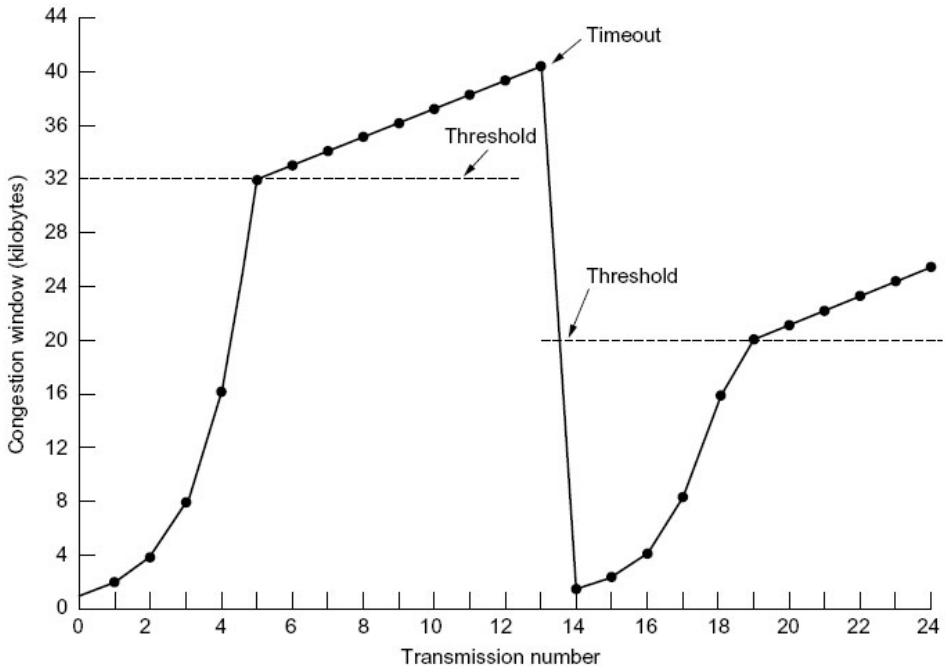


TCP ima "žagasto obliko" hitrosti pošiljanja

TCP nadzor zamašitev

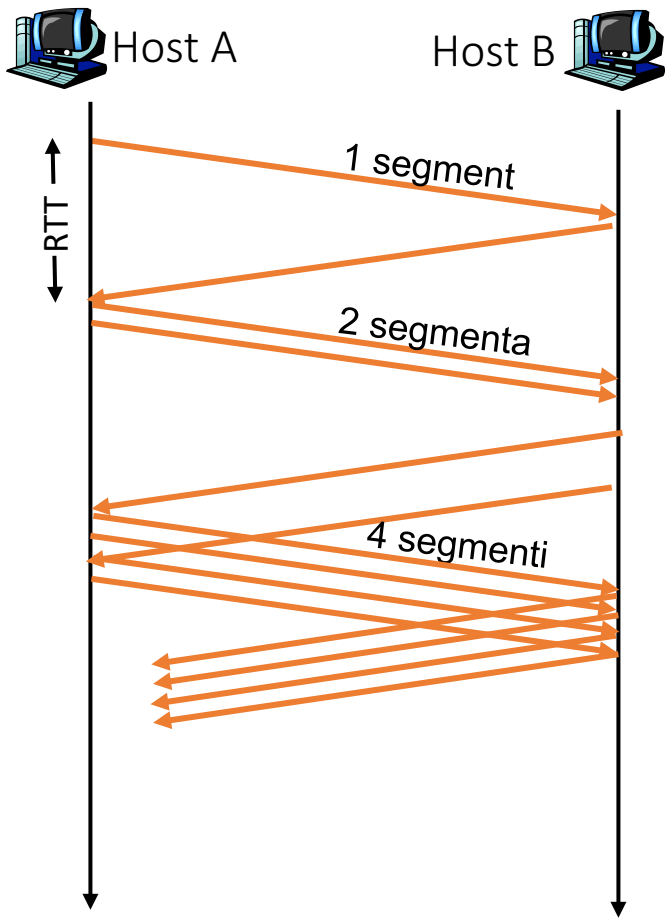
- okno **rwnd** (*receive window*) smo že spoznali (omejitev količine nepotrjenih podatkov za kontrolo pretoka)
- za nadzor zamašitev uporabljamo okno **cwnd** (*congestion window*). TCP torej pošilja s hitrostjo, ki ustreza **min(rwnd, cwnd)**

- možni dogodki:
  - POZITIVEN:
    prejem ACK: povečuj cwnd
    - eksponentno (x2, **počasni začetek**, *slow start*) ali
    - linearno (+1, **izogibanje zamašitvam**, *congestion avoidance*)
  - NEGATIVEN:
    potek časovnega intervala (segment se izgubi): zmanjšaj cwnd na 1



Počasen začetek (*Slow Start*)

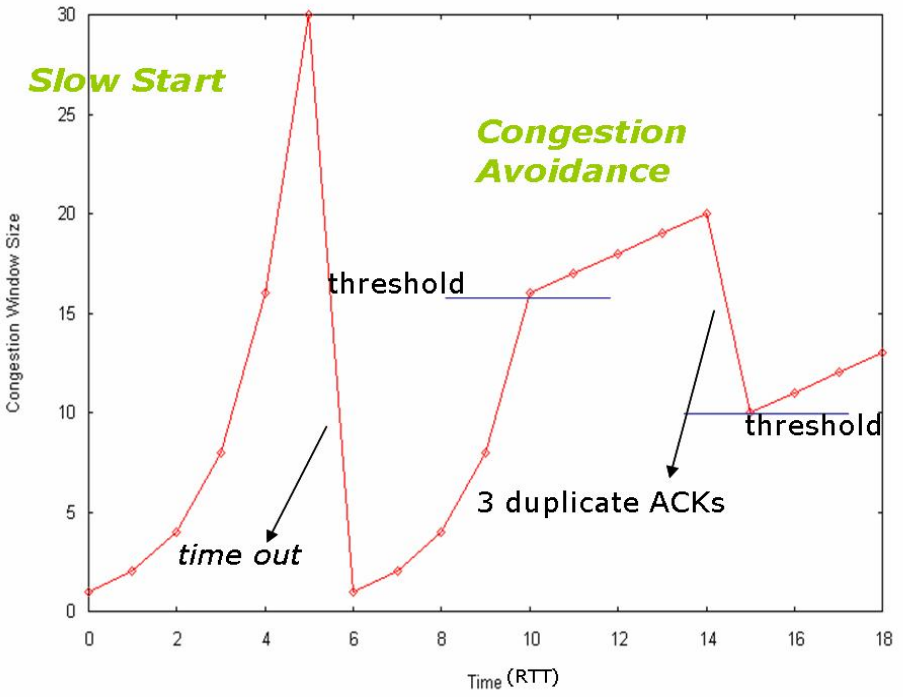
1. ob vzpostavitvi povezave:
   velja **cwnd = 1** segment
2. hitrost povečuj eksponentno, tako da za vsak prejeti ACK:
   cwnd <- cwnd*2
3. ko pride do prve izgube, se ustavi in si zapomni **PRAG** (polovica trenutnega cwnd, ko pride do zamašitve) ter nastavi cwnd=1
4. izvajaj počasen začetek od koraka 1. Ko prideš do vrednosti PRAG, preidi v način **izogibanja zamašitvam**.



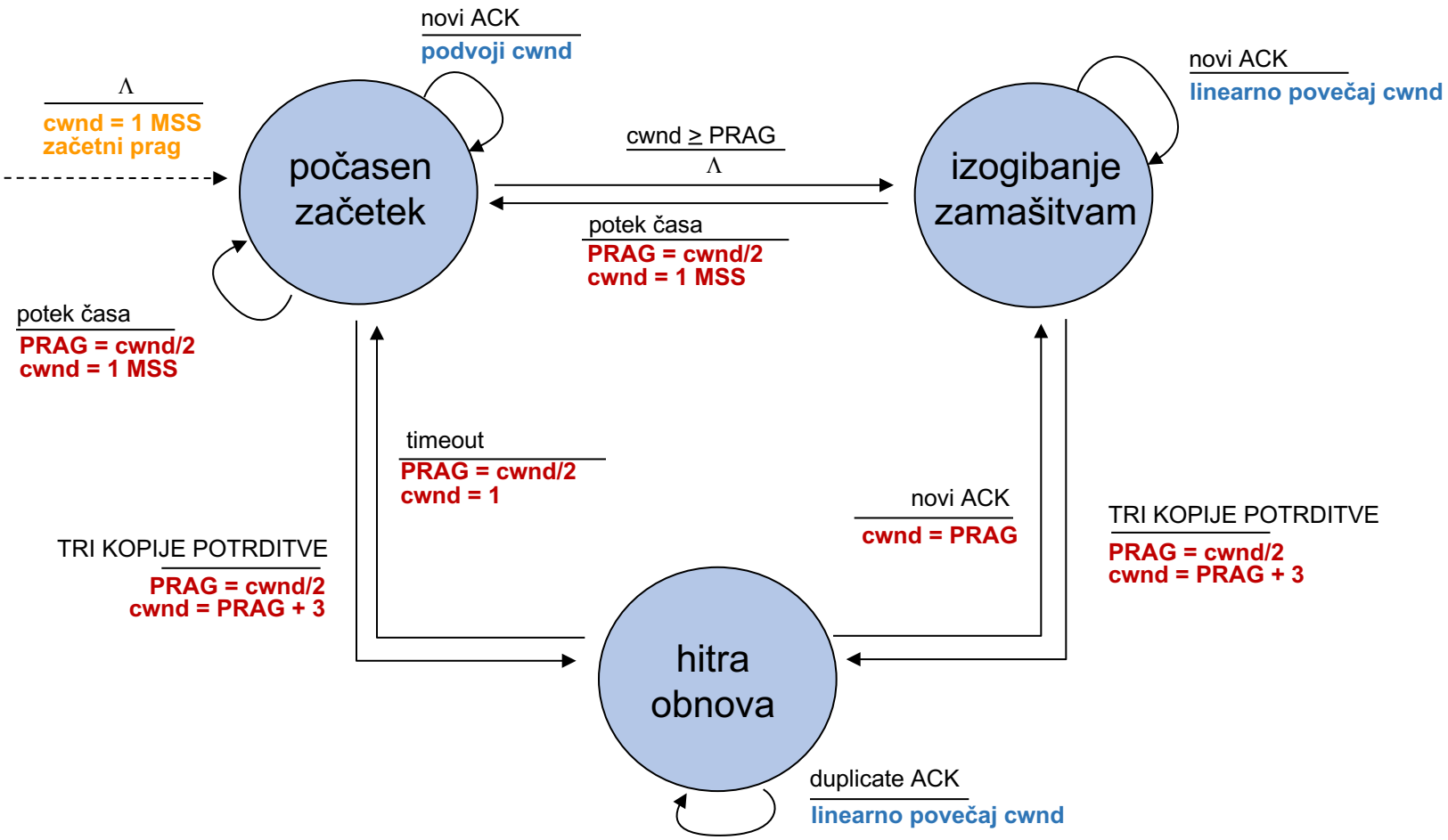
Izogibanje zamašitvam (*Congestion Avoidance*)

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

- poznamo tudi način **hitre obnove** (*fast recovery*), v katero preideta počasen začetek in izogibanje zamašitvam ob prejemu 3 ponovljenih ACK (prepolovi cwnd + 3).

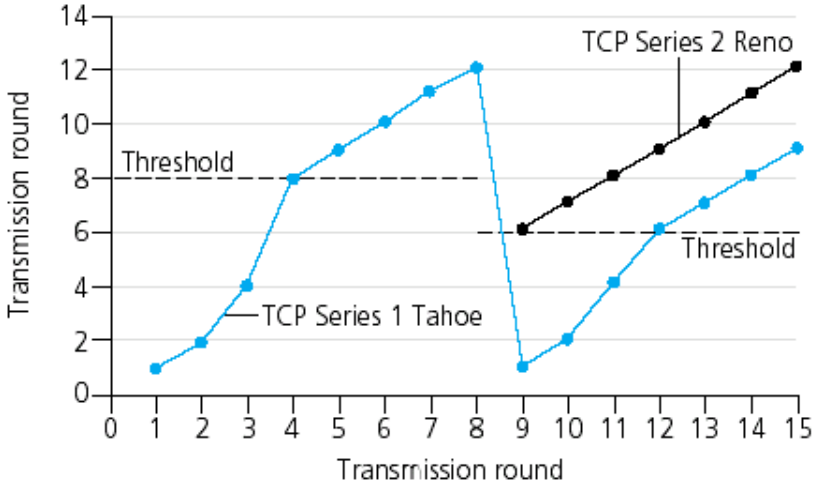


Končni avtomat za TCP nadzor zamašitev

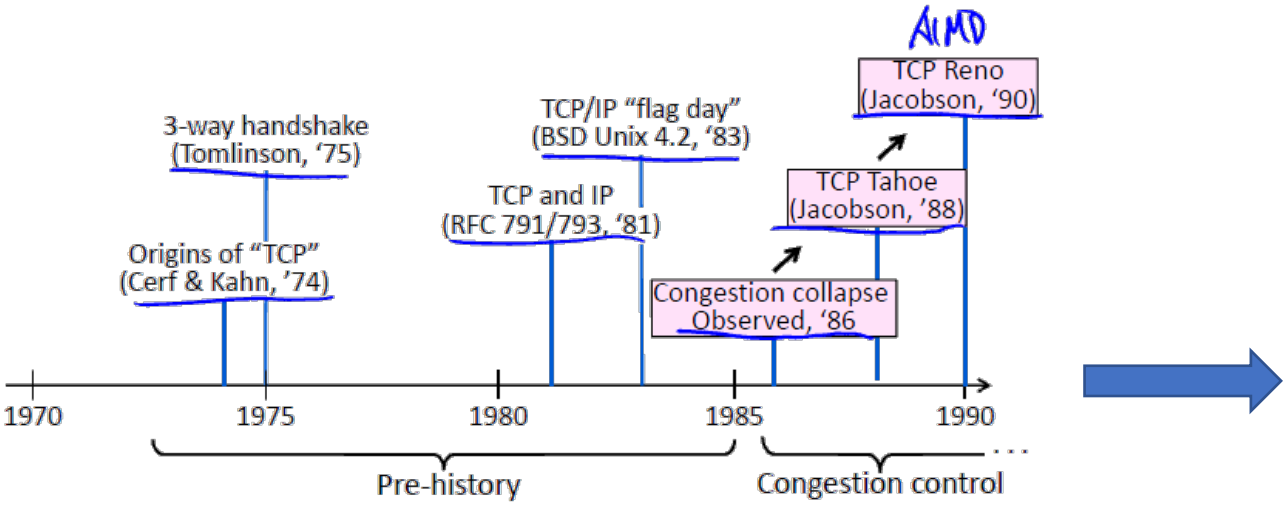


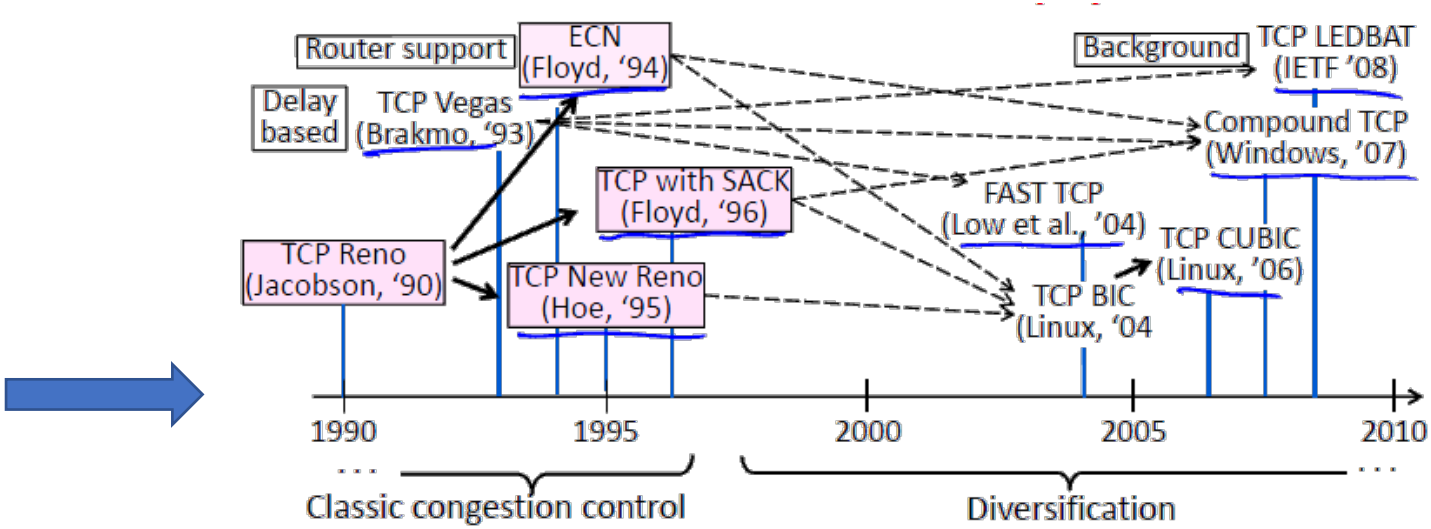
Razvoj nadzora zamašitev skozi različice TCP

1. **TCP Tahoe**: osnovna verzija (uporablja samo *počasen začetek* in *izogibanje zamašitvam*), 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 zamašitve in linearno zmanjševanje hitrosti pošiljanja ob zamašitvah

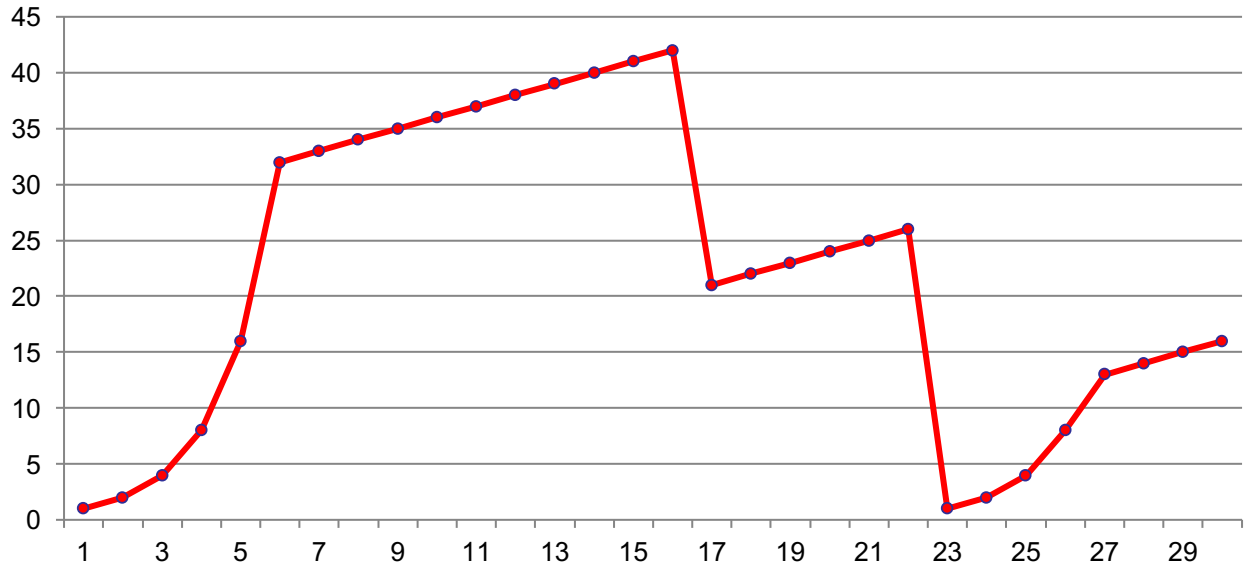


Zgodovina razvoja TCP





Primer: analiza delovanja TCP Reno

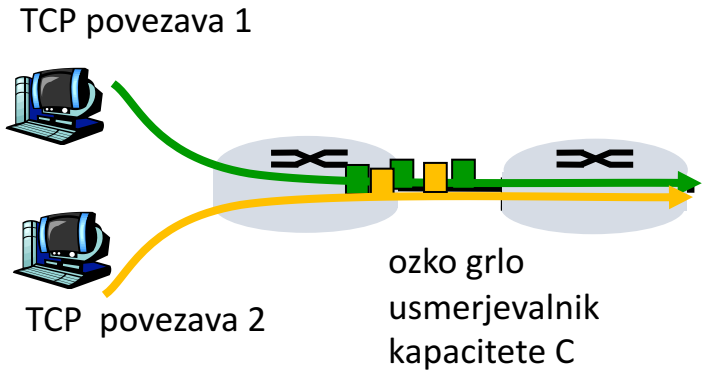


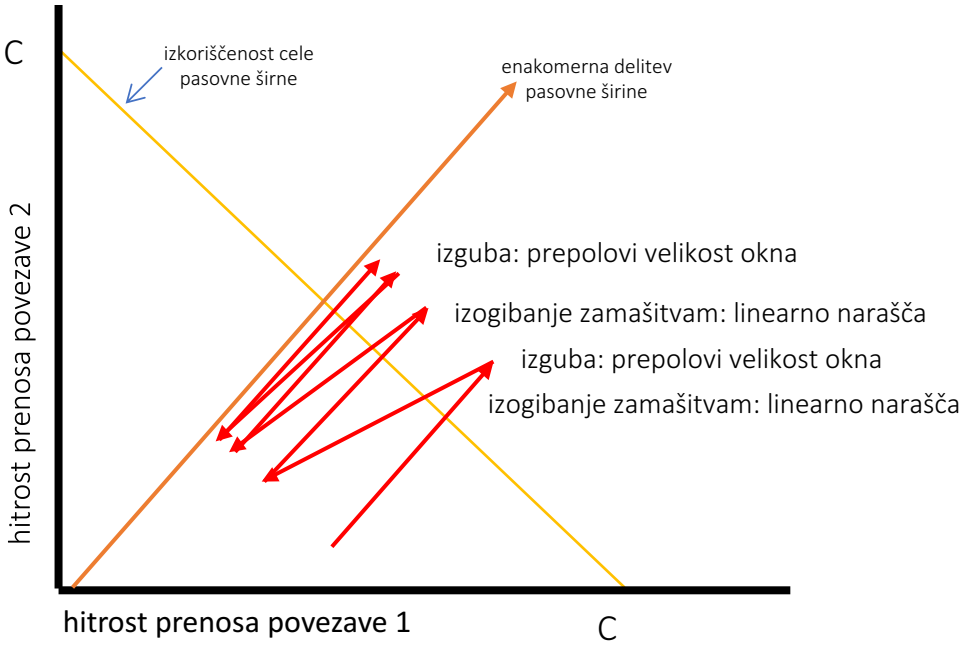
Primeri vprašanj za analizo delovanja protokola:

1. Kdaj se je izvajal počasen začetek in kdaj izogibanje zamašitvam?
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 zamašitev skonvergira v sredinsko točko grafa)





Pravičnost TCP in UDP

- Uporaba TCP in UDP:
  - TCP: SMTP, Telnet, HTTP, FTP, ...
  - UDP ali TCP: SIP, pretočne aplikacije, ...
  - tipično UDP: DNS, SNMP, RIP (usmerjanje), telefon (zaprti protokoli) itd.

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