Univerza v Ljubljani

Fakulteta
za računalništvo
in informatiko

Transportna plast

© Mojca Ciglarič

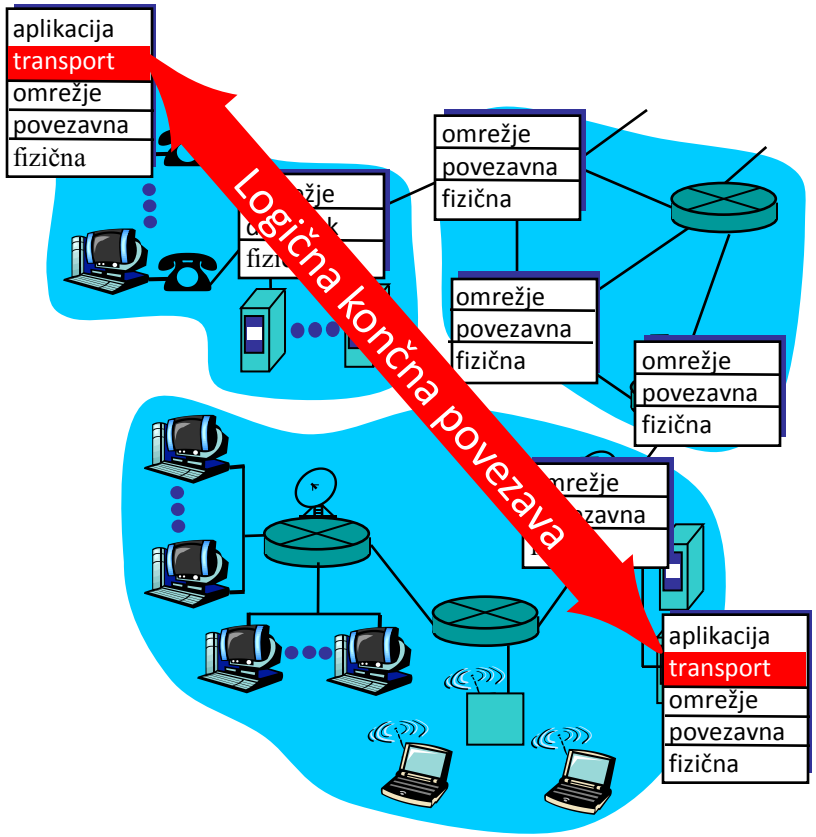
Pregled

- Storitve transportne plasti
- (De-)multipleksiranje
- UDP
- TCP in zanesljiv prenos
- TCP in nadzor zamašitev (zasičenje - congestion control)

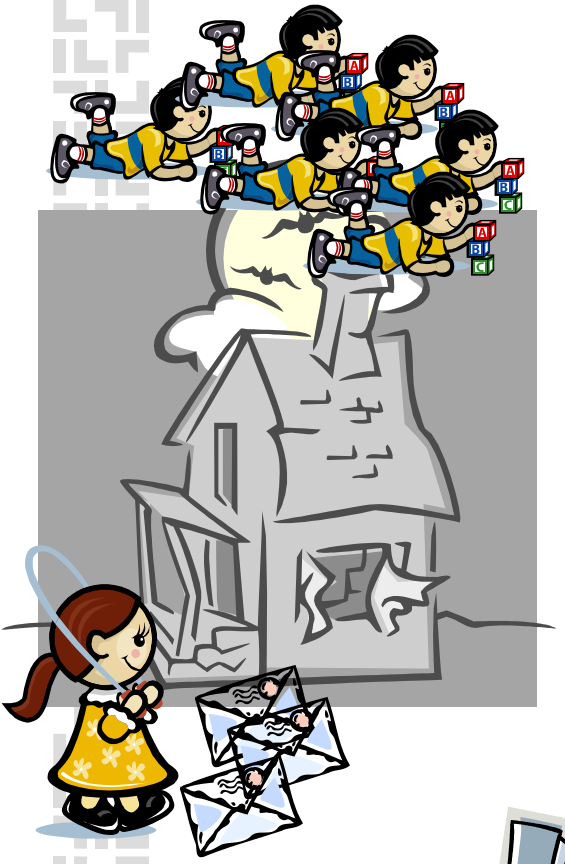
Transportne storitve in protokoli

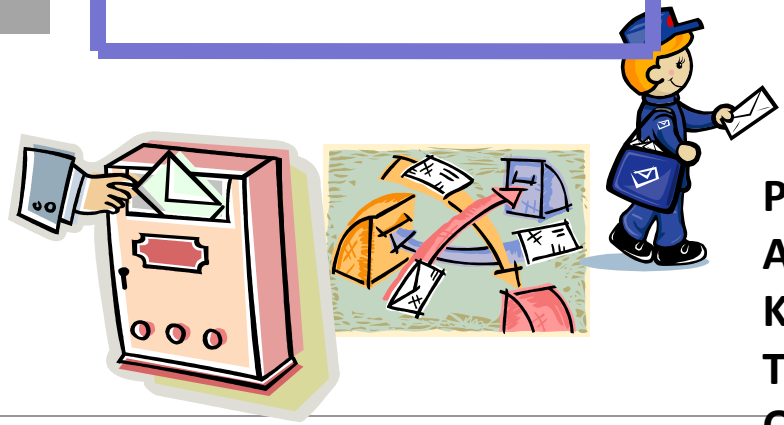
- Logična komunikacija med aplikacijskimi procesi
- Transportni procesi tečejo v KONČNIH sistemih
- Naloga:
  - Sprejem sporočila od aplikacije
  - Sporočilo -> segmenti -> omrežna plast
  - Sprejem PDUjev od omrežne plasti
  - Sestavljenje segmentov v sporočilo
  - Predaja aplikacijski plasti
- Internet: ni zagotovitve pasovne širine ali zakasnitve
  - TCP (vzp. povezave, kontrola pretoka in zamašitev)
  - UDP (best effort)

Logična povezava



Analogija

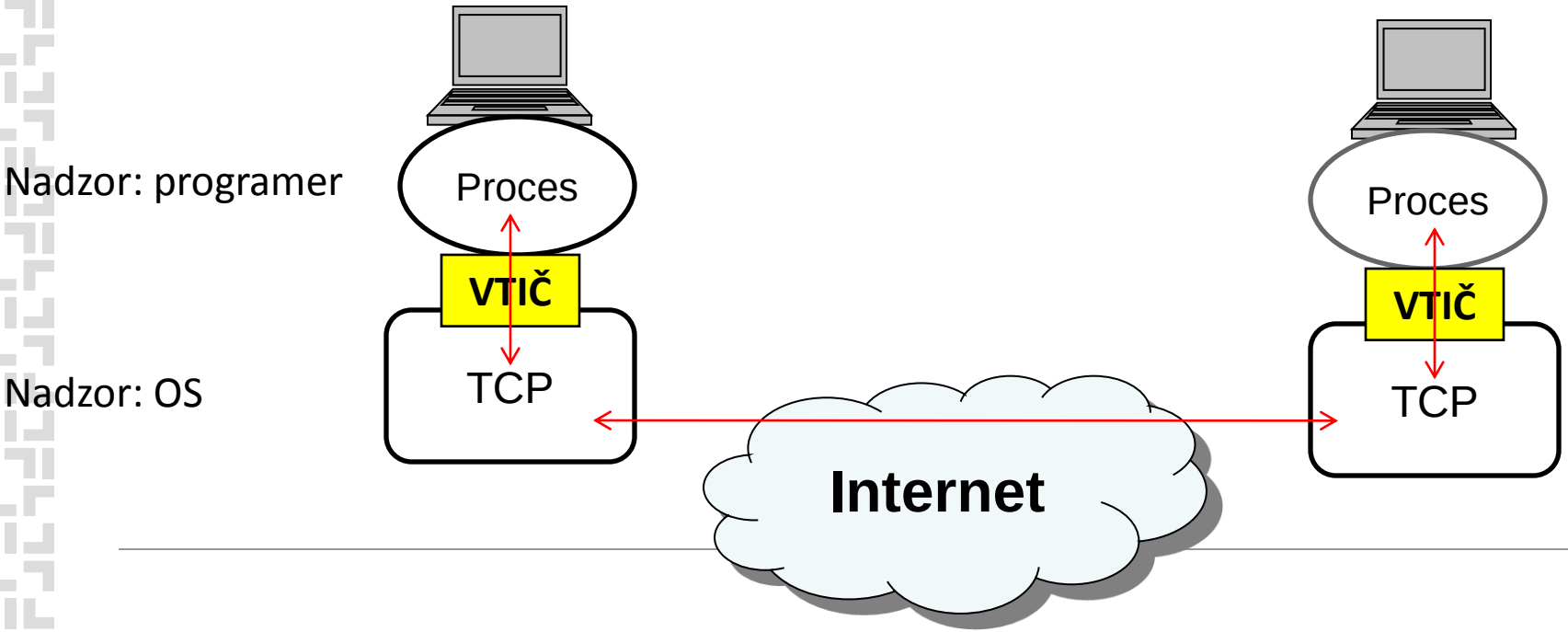




Proces = otrok
Apl. sporočilo = pismo
Končni rač. = hiša
Transportni protokol = Ana in Eva
Omrežna plast = storitev pošte

Vtiči (socket)

- Vtič je vstopna točka v proces.
- Vtič je **vmesnik** (API) med aplikacijsko in transportno plastjo.



Kako nasloviti proces na drugi strani?

- **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 port*), npr.
  - Spletni strežnik: 80
  - Poštni strežnik SMTP: 25
  - Imenski strežnik: 53
  - IRC strežnik: 194
- Več: **`www.iana.org`**

Kaj potrebuje aplikacija?

- Kako izbrati transportni protokol?
  - Kaj nudi TCP in kaj UDP?
  - Vlak ali letalo? Avtobus ali kolo?
- Bistveno
  - Zanesljiv prenos podatkov ali tolerira izgubo?
  - Zagotovljeno pasovno širino? (aplikacije, občutljive na pasovno širino / elastične aplikacije)
  - Čas: omejena ali neomejena zakasnitev

Storitve TCP-ja

- Povezana storitev *(povezavno usmerjena, connection-oriented)*
  - 1.faza – rokovanje - handshaking: vzpostavljanje TCP povezave (kontrolna sporočila)
  - 2.faza – prenos aplikacijskih sporočil, popolnoma dvosmerna povezava
  - 3.faza – rušenje povezave
- Zanesljiv prenos: brez napak, v pravem zaporedju.
- Nadzor zamašitev *(congestion control)*
- NE zagotavlja kapacitete ali zg. meje zakasnitve.

Storitve UDP-ja

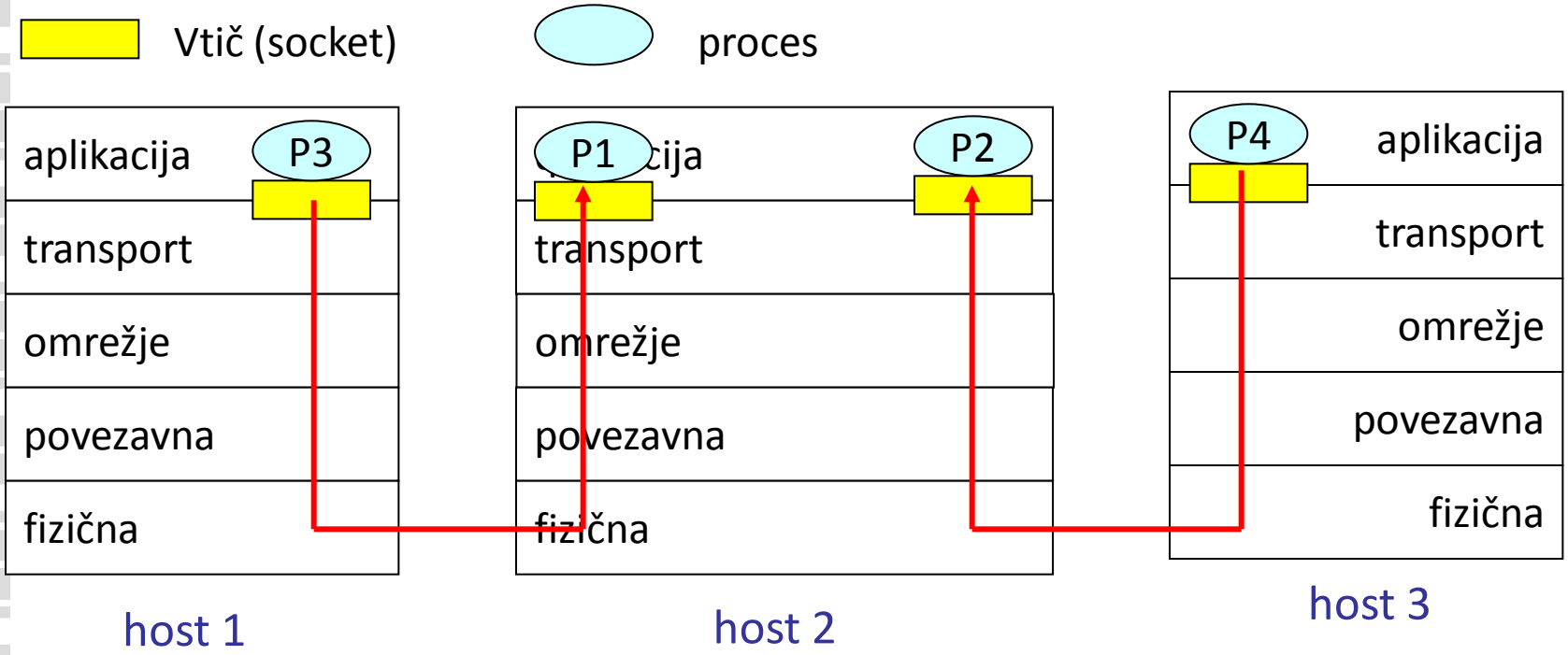
- Hiter, učinkovit, lahek, minimalističen
- Nepovezaven – brez “rokovanja”
- Ni garancije dostave, ne zagotavlja vrstnega reda
- Nima nadzora zamašitev

Uporaba

- **TCP**: SMTP, Telnet, HTTP, FTP, ...
- **UDP ali TCP**: SIP, pretočne aplikacije,..
- **Tipično UDP**: DNS, SNMP, RIP (usmerjanje), telefon (zaprti protokoli)...

(De-)multipleksiranje – kaj je?

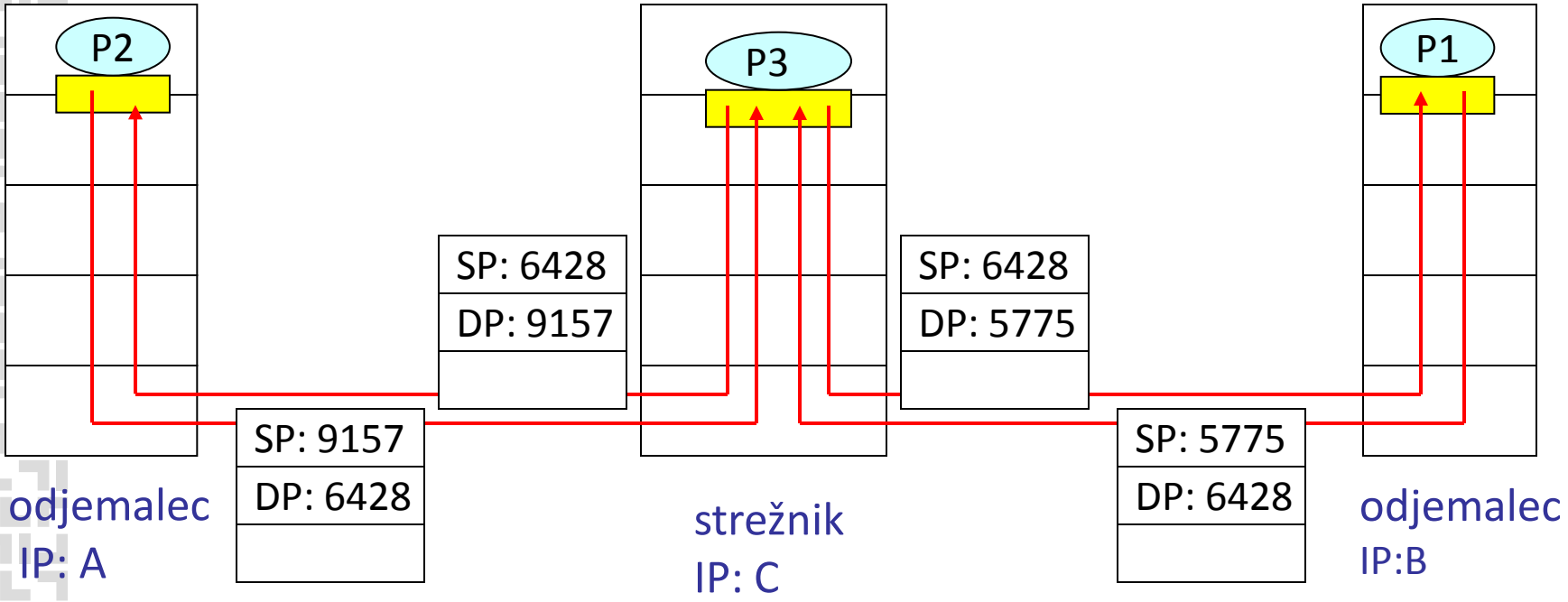
- Pošiljatelj: pobira iz več vtičev, doda glavo
- Sprejemnik: segmente razdeli ustreznim vtičem



(De-)multipleksiranje – kako?

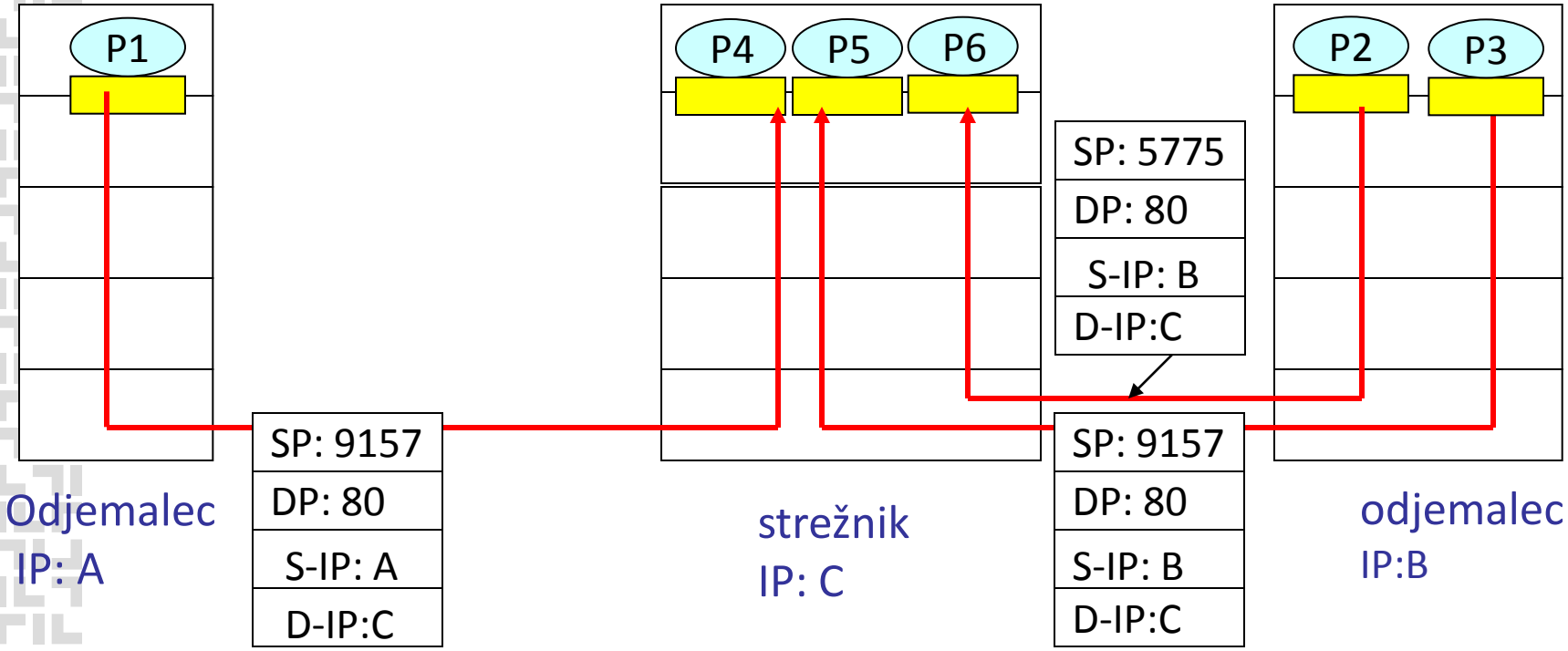
- IP datagram: IP naslova izvora in ponora
  - nosi en transportni segment
- Transportni segment: št. vrat izvora in ponora
- Ciljni računalnik:
  - IP naslov + številka vrat → ugotovi pravi socket
- UDP vtič določen s parom: (dest IP, dest port)
  - Lahko sprejema iz različnih IP naslovov ali vrat
- TCP vtič: (source IP, source port, dest IP, dest port)
  - En proces ima lahko več vtičev

Nepovezavno demux.-primer

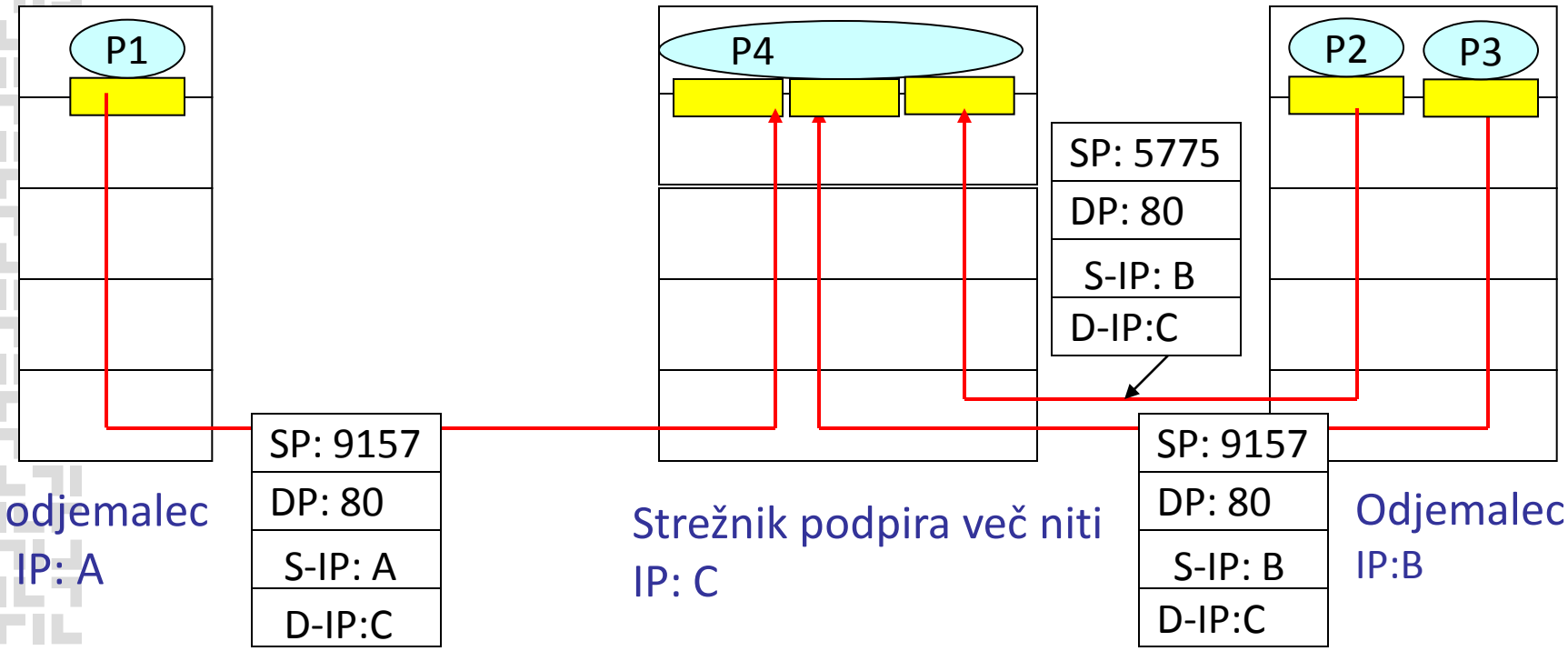


SP = source port (predstavlja naslov za odgovor)
DP = destination port

Povezavno demux.-primer



Povezavno demux.- primer z nitmi



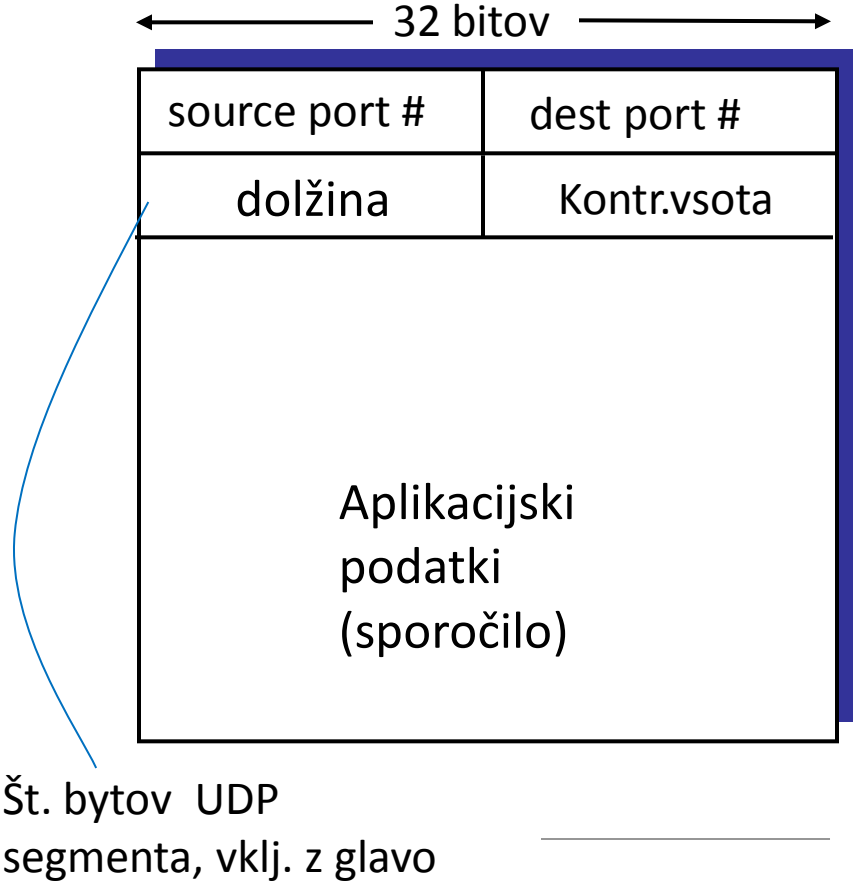
UDP – User Datagram Protocol

- RFC 768 - minimalna funkcionalnost
- Storitev po najboljših močeh (best effort):
  - Možno izgubljanje, napačen vrstni red
- Nepovezavni: ni rokovanja, vsak datagram potuje posebej (UDP segment = datagram)
- Prednosti:
  - Ni zakasnitve rokovanja
  - Preprost, ni treba vzdrževati stanja (obe strani)
  - Majhna glava
  - Ni kontrole zamašitev – lahko hitro oddaja

UDP

- Uporaba: večpredstavnost
  - Tolerira izgube
  - Hitrost!
- DNS, SNMP
- Zanesljiv prenos:
  - Zanesljivost se zagotovi na apl. plasti

UDP: format datagrama



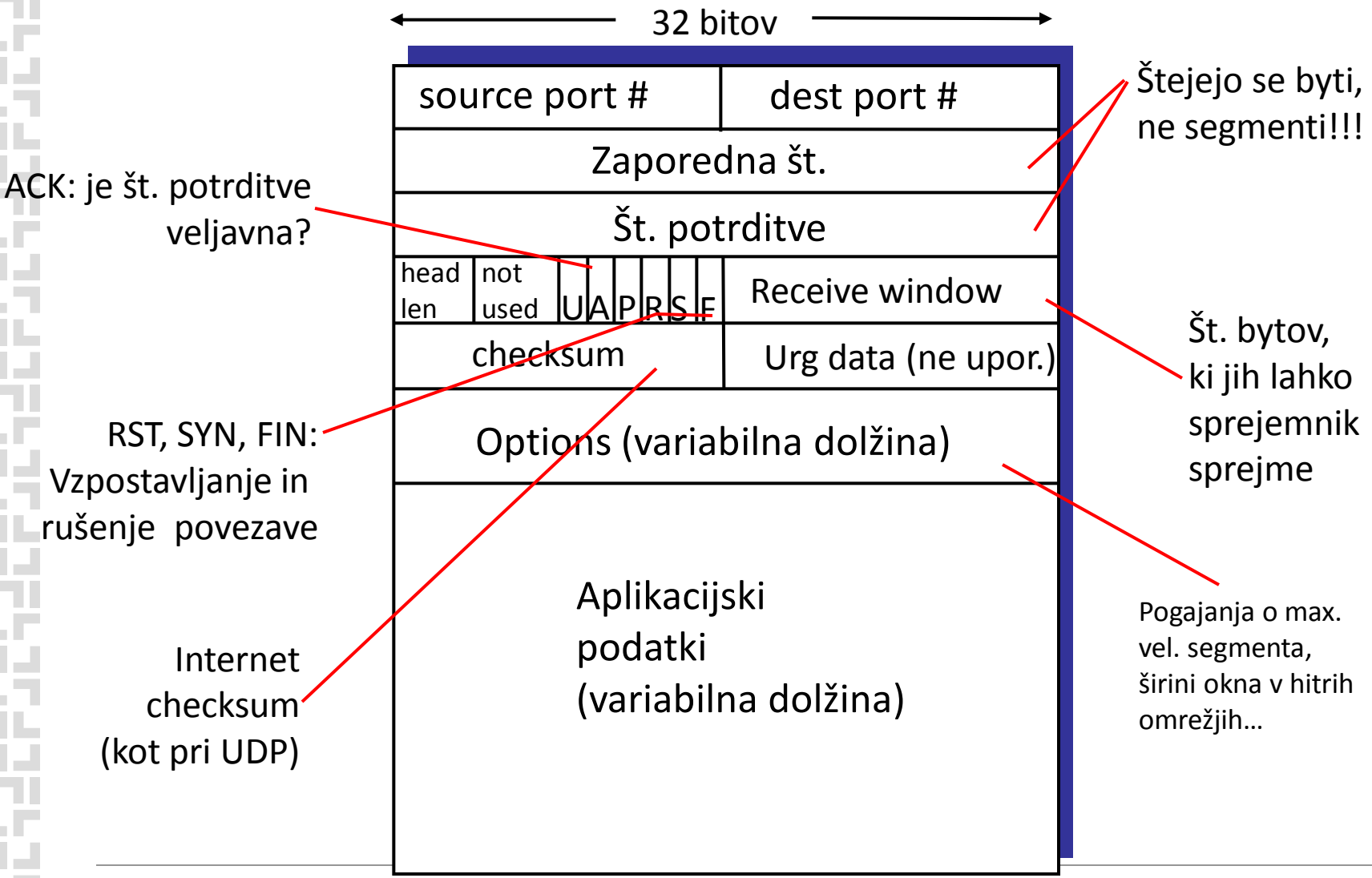
Zanesljiv prenos podatkov (potrjevanje)

- Zanesljiv kanal (ni pokvarjenih, izgubljenih pak.)
- Nezanesljiv kanal - ključni principi:
  - ACK, NACK, detekcija napak
  - Izguba, okvara ACK, NACK
  - Zap. št. paketov, duplikati
  - Časovne kontrole
  - Stop and wait (sprotno)
  - Go-back-N, selective repeat: tekoče
  - NACK-free protokol: namesto NACK pošlje ACK za zadnjega pravilnega (ACK5, ACK5 velja kot NACK6)

TCP

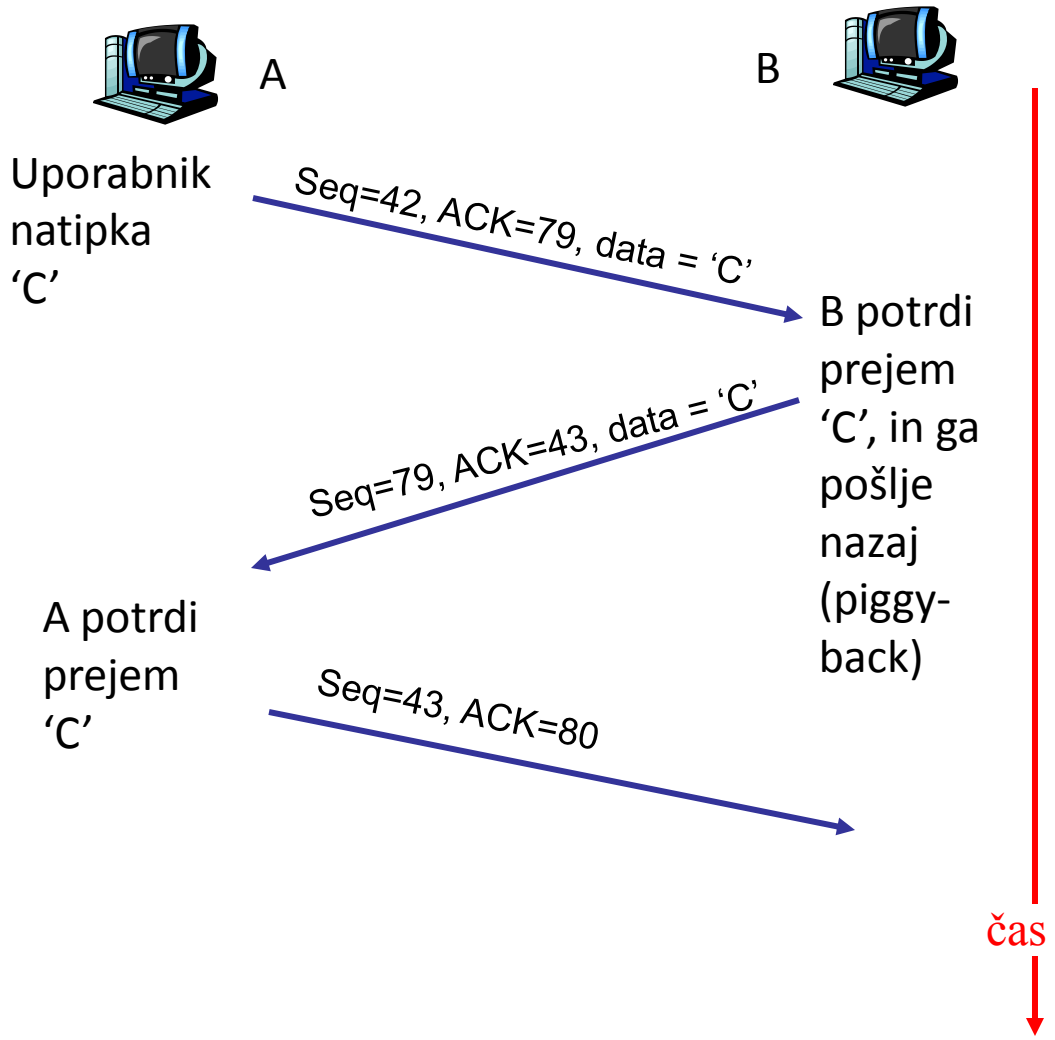
- RFC 793, 1122, 1323, 2018, 2581
- Lastnosti
  - En sprejemnik, en oddajnik
  - Zanesljiv, urejen tok
  - Sprejemni in oddajni “TCP bufferji”
  - “cev” od izvora do ponora, nadzor pretoka in zamašitev
  - Oddajnik ne “zasuje” sprejemnika
  - Full duplex, max. velikost segmenta
  - Povezavno usmerjen (rokovanje)

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?
Št. bytov, ki jih lahko sprejemnik sprejme
RST, SYN, FIN: Vzpostavljanje in rušenje povezave
Pogajanja o max. vel. segmenta, širini okna v hitrih omrežjih…
Internet checksum (kot pri UDP)

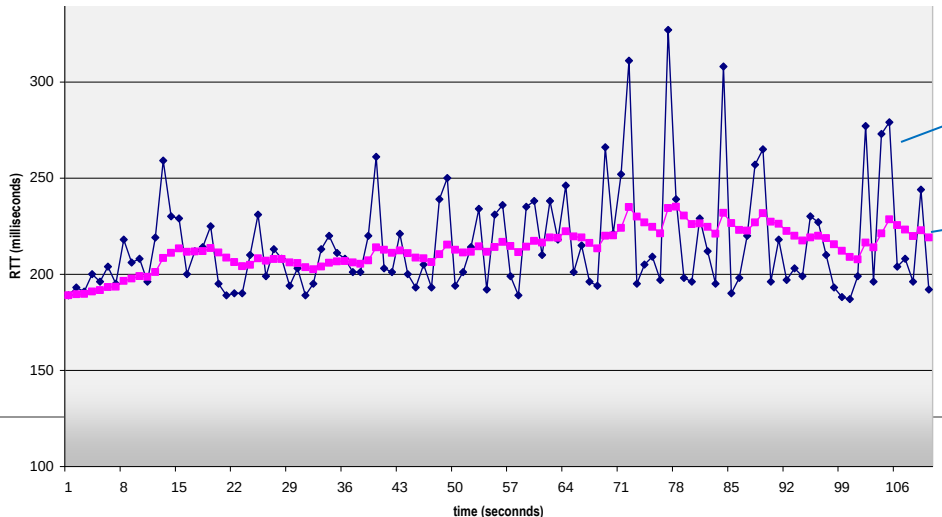
Primer

- Delček telnet seje: strežnik vrača vtipkane črke nazaj, šele potem jih uporabnik vidi na zaslonu.
- Kaj je **čas vrnitve** (RTT- round-trip time)?



Čas vrnitve

- Čas vrnitve - RTT (Round trip time)
- Interval za časovno kontrolo > povprečni RTT !
- TCP meri čas do ACK (SampleRTT) in računa gibajoče povprečje.
- **`Ocena RTT' = OcenaRTT(1-α) + α*SampleRTT`**



α =0.125 (tipično)

Interval za časovno kontrolo

- Če čas vrnitve bolj niha, naj bo večja ‘rezerva’.

```
OdmikRTT' = (1 - β) OdmikRTT +
                    β * |SampleRTT – OcenaRTT|
```

β = 0.25 (tipično)

```
TimeoutInterval = OcenaRTT + 4*OdmikRTT
```

- **Fast retransmit**: ponovna oddaja segmenta preden poteče časovna kontrola, če dobi 3x ACK za isti segment.

TCP: zanesljiv prenos

ACK (posredno), časovne kontrole, tekoče pošiljanje

osnova = začetna zap. št. ; *nasl* = začetna zap.št.

Če sprejme podatke od aplikacije:

- Naredi TCP segment *nasl*, sproži timer
- Segment preda IP-ju, *nasl* = *nasl* + dolžina podatkov

Če poteče timer za segment y

- Ponovno odda segment y
- sproži timer

Če sprejme potrditev za segment y

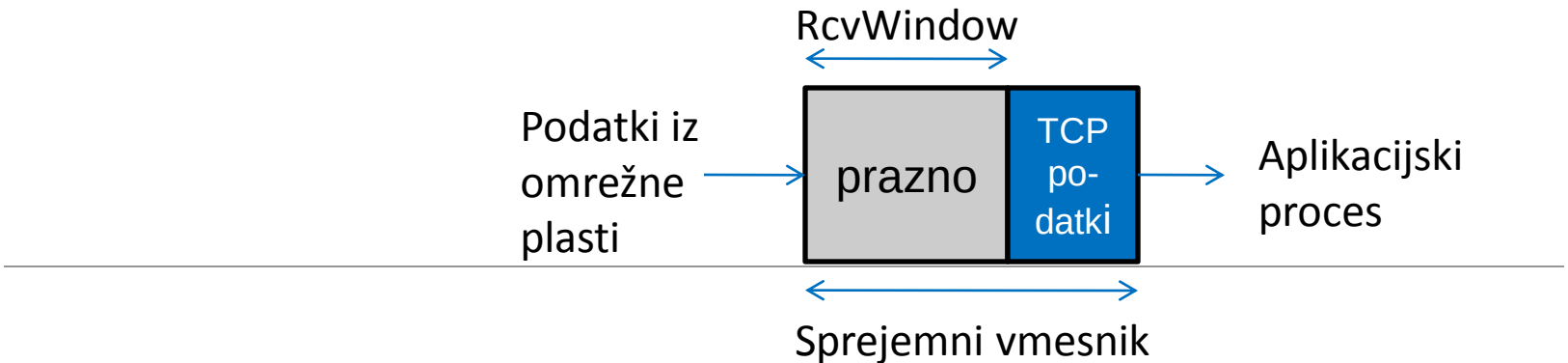
- Če y > *osnova*: /* potrditev zaporedja do vklj.y */
  Zbriši timer za segmente y < *nasl*; *osnova* = y
- Sicer /* duplikat že prejete potrditve */
  poveča števec duplikatov ACK za y; če je ta že = 3, ponovi segment y in ponastavi timer zanj
  /* to je TCP fast retransmit*/

TCP potrditve (RFC 1122, 2581)

Dogodek pri sprejemniku	Odziv sprejemnika
Sprejem naslednjega segmenta, prejšnji že potrjen	Zakasnjen ACK. Po max. 500 ms potrdi, če ni nasl. sprejema.
Isto, a prejšnji nepotrjen.	Kumulativni ACK – potrdi oba.
Sprejem segmenta s previsoko št. (vrzel!)	Takoj potrdi zadnji v zaporedju sprejeti segment (pošlje duplikat).
Sprejem segmenta z najnižjo številko iz vrzeli (polnjenje vrzeli)	Takoj potrdi segment.

TCP: kontrola pretoka

- Aplikacija bere iz sprejemnega vmesnika (receive buffer) – lahko počasi.
- Prejemnik:
  - **`RcvWindow`** polje v glavi segmenta: sem vpiše količino praznega prosotora v spr. vmesniku
- Pošiljatelj nastavi širino okna na **`RcvWindow`**



TCP: vzpostavljanje povezave

Trojno rokovanje (three-way handshake)

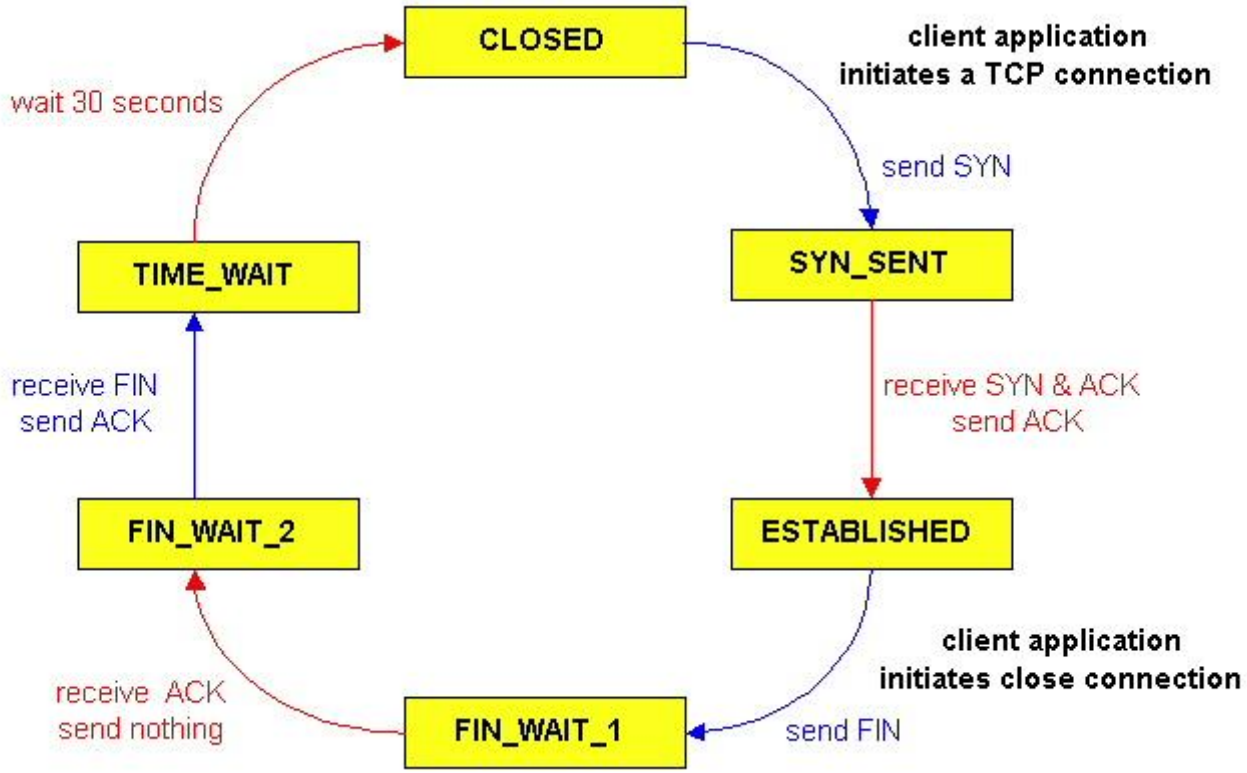
1. Odjemalec pošlje TCP SYN segment
   - Začetna št. odj. segmenta, brez podatkov
2. Strežnik vrne SYNACK segment
   - Začetna št. str. segmenta; alocira buffer
3. Odjemalec vrne ACK, lahko že s podatki

TCP: rušenje povezave

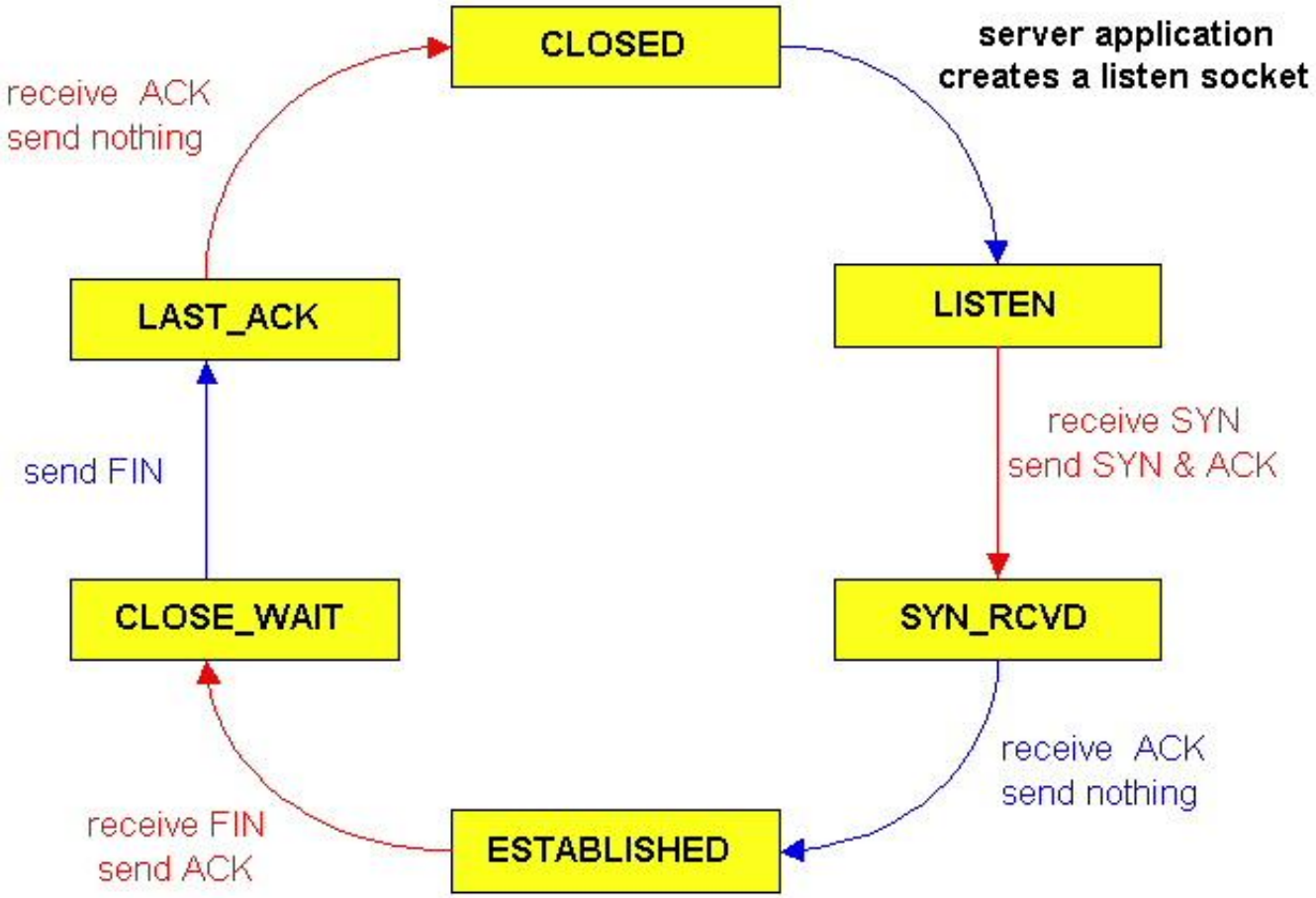
Odjemalec zapre socket:

1. Odjemalec pošlje TCP FIN segment.
2. Strežnik potrdi z ACK, zapre povezavo, pošlje FIN.
3. Odjemalec potrdi FIN z ACK.
   - Čaka kratek čas; če sprejme FIN, potrdi z ACK.
4. Strežnik sprejme ACK, končano.

Življenski cikel odjemalca



Življenjski cikel strežnika

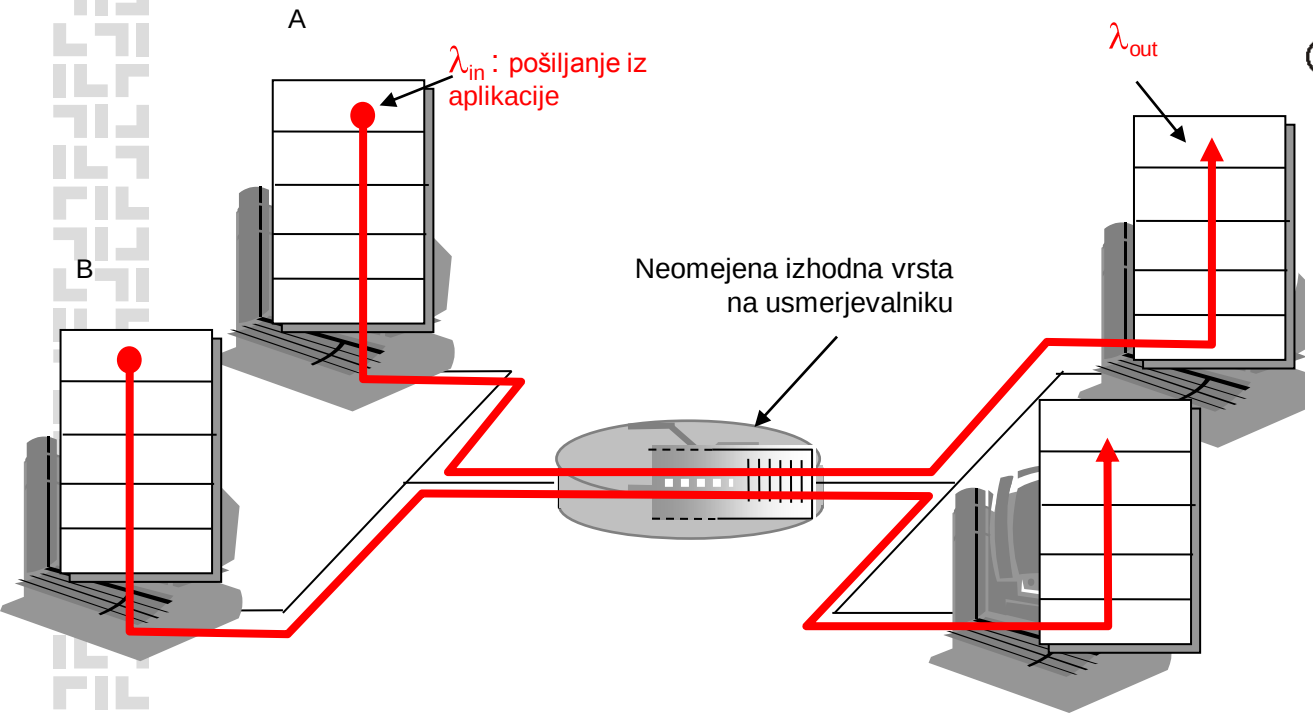


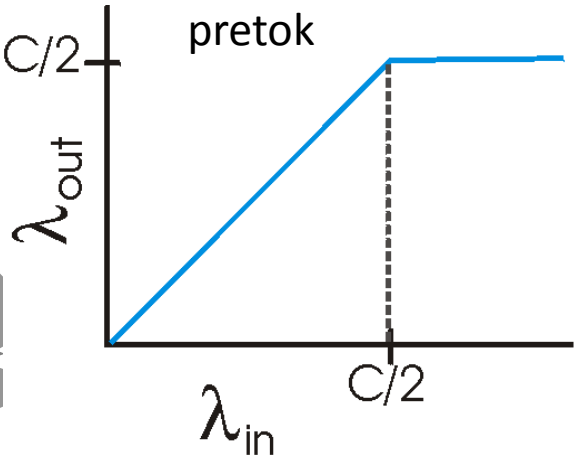
Nadzor zamašitev (congestion)

- Ni isto kot nadzor pretoka!
- Zamašitev: preveč virov naenkrat pošilja preveč podatkov (prehitro) za dano omrežje.
- Posledica:
  - izguba segmentov (prelivi),
  - velike zakasnitve (dolge vrste na usmerjevalnikih)

Zamašitev – primer 1

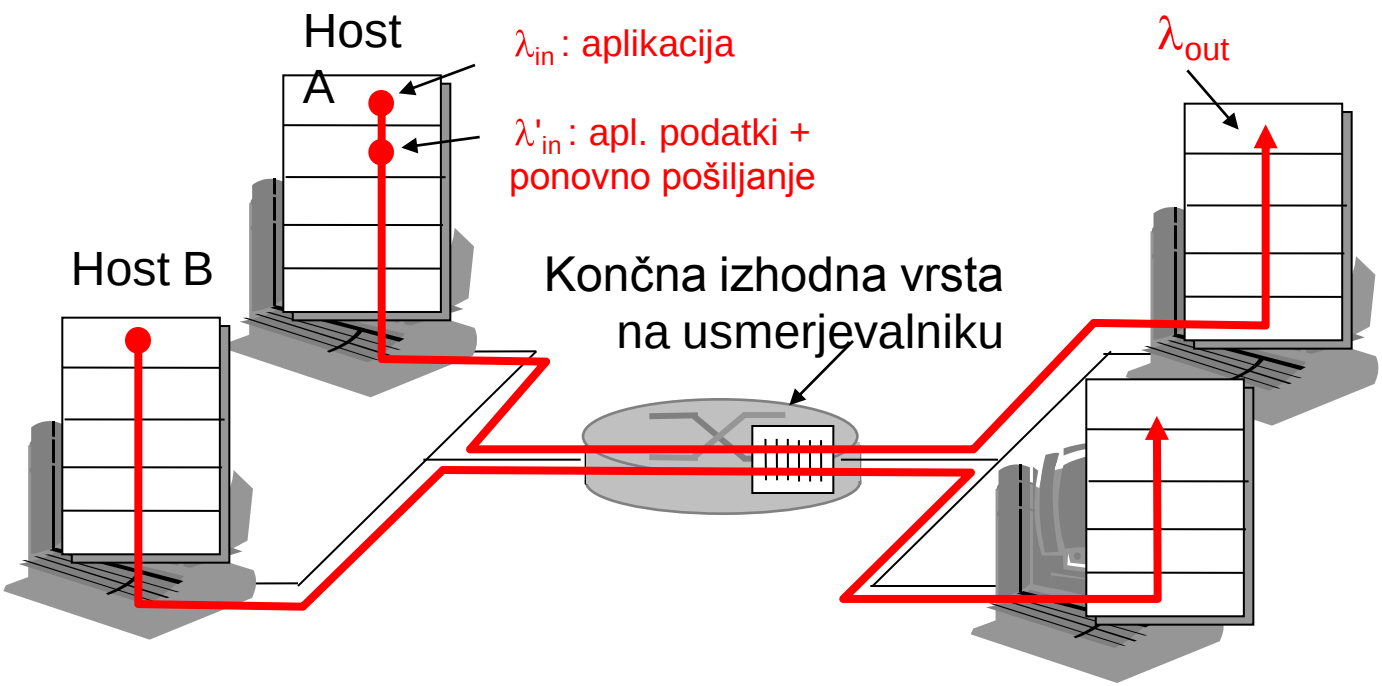
- Dva pošiljatelja, neskončna vrsta





Zamašitev – primer 2

- Končna vrsta in ponovna pošiljanja segmentov



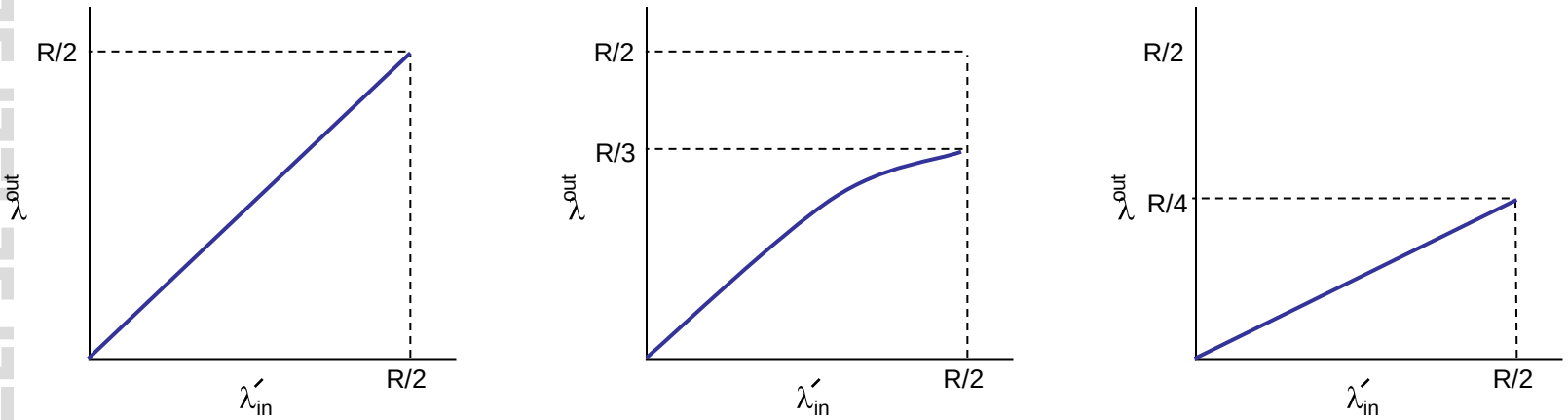
Zamašitev – primer 2

a. Če bi vedeli, kdaj je prostor v vrsti, ne bi bilo izgub: segment oddamo le, ko je prostor (ni možno v praksi)

b. Dogajajo se izgube paketov in ponovna pošiljanja

c. ponovna pošiljanja tudi zaradi velikih zakasnitev

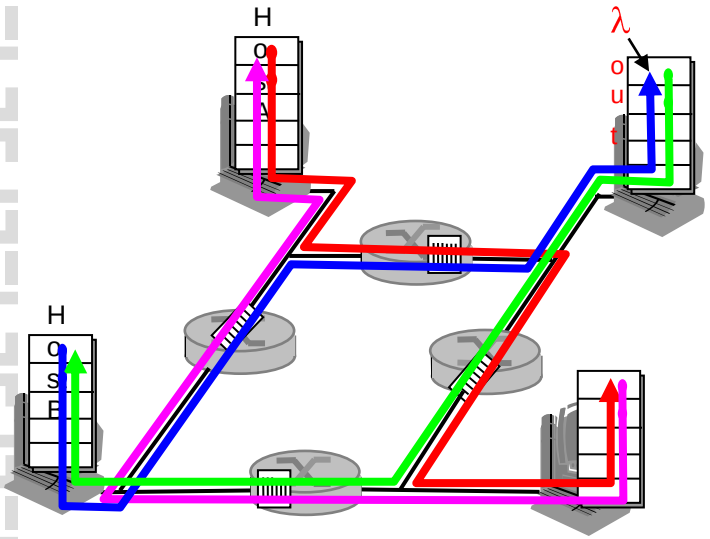
Več dela omrežja za manjši učinek. Nepotrebne ponovitve.

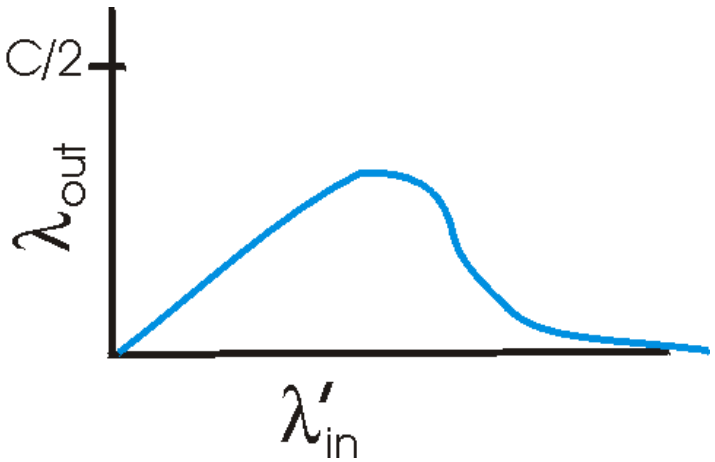


a. b. c.

Zamašitev – primer 3

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





Nadzor zamašitev - pristopi

1. TCP: Odvisno le od **končnih sistemov**: opazijo izgubo ali zakasnitve.
2. Lahko pa pomaga tudi **omrežje**:
   - Usmerjevalnik nastavi kak bit (SNA, DECnet, ATM)
   - Usmerjevalnik sproči sprejemljivo hitrost oddajanja

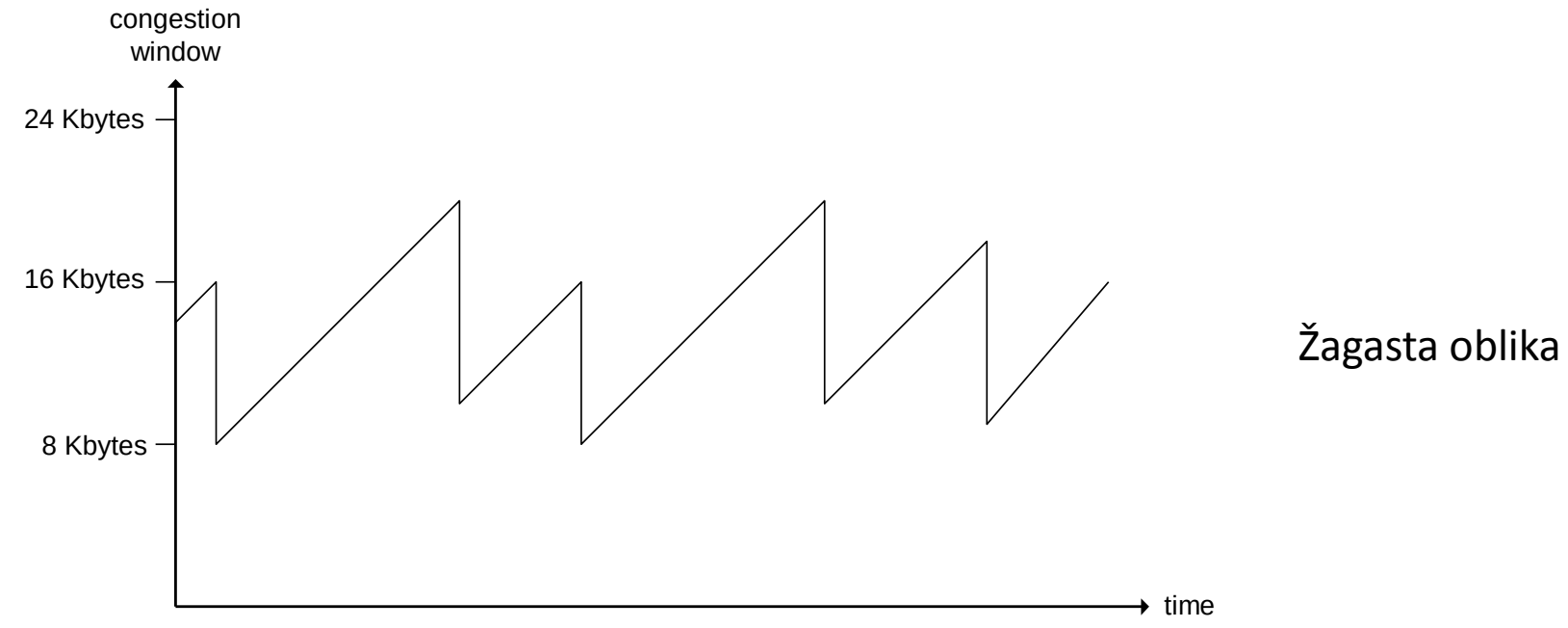
- Primer ATM: kontrolna RM celica (resource mgmt.)
  - Pošiljatelj jih oddaja med podatkovnimi
  - ATM stikalo lahko nastavi NI (no increase – ne povečuj prometa) ali CI (congestion indication – zmanjšaj) bit
  - Prejemnik vrne RM celice nespremenjene oddajniku

TCP: nadzor zamašitev

- Dinamični vrednosti **`CongWin`** (zamašitveno okno) in **`treshold`** – prag.
- Hitrost pošiljanja ≈ **`CongWin/RTT`** bytov/s
- Pošiljatelj: izguba (č.k. ali 3 duplikati ACK) → zmanjša **`CongWin`**

TCP AIMD

- Additive Increase: CongWin se vsak RTT poveča za 1 max. velikost segmenta, če ni napak.
- Multiplicative Decrease: ob izgubi (3 duplikati ACK) zamašitveno okno prepolovimo.



Žagasta oblika

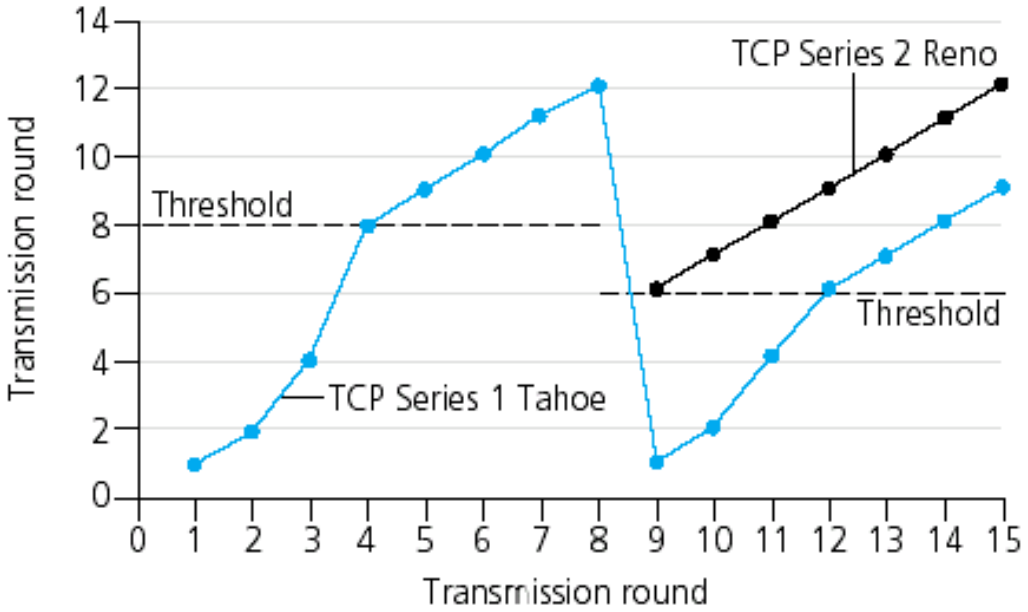
Počasen začetek – TCP Slow Start

- Ob vzpostavitvi povezave: **`CongWin`** = 1 segment
  - Primer: če MSS = 500 bytov in RTT = 100 ms, potem je začetna hitrost 39 kbps.
- Povečuj hitrost eksponentno (**`CongWin`***2 vsak RTT – ob prejemu ACK) do prve izgube (tu nastavi **prag**)

Nadaljnja izboljšava:

- Po treh duplikatih ACK razpolovi **`CongWin`**, okno nato povečuj linearno (za 1) – cong. avoidance
- Po timeoutu postavi **`CongWin`** = 1 , nato naj raste eksponentno do praga (slow start), nato linearno.

TCP različice

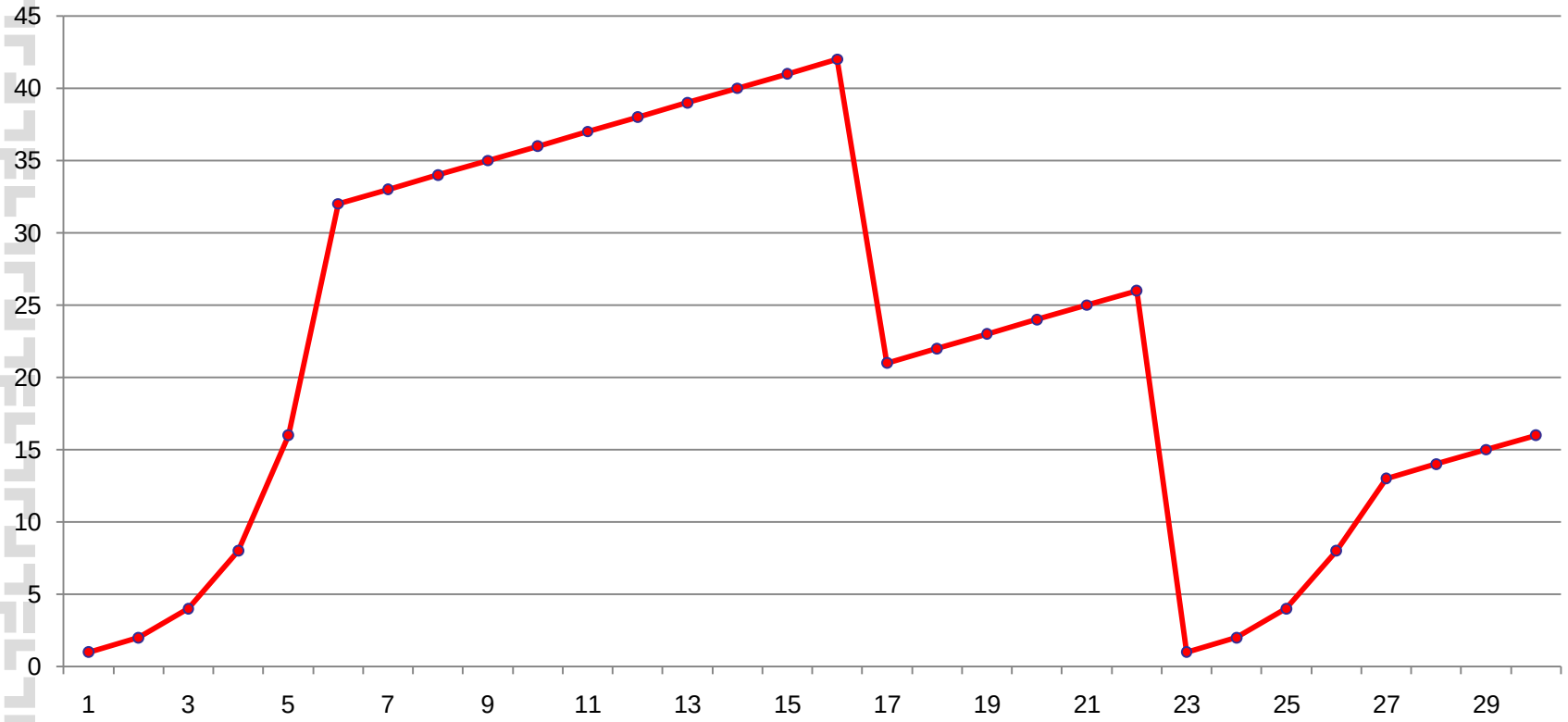


- TCP Tahoe: osnovna verzija
- TCP Reno: ni toliko čakanja po č.k. – *fast recovery:* preskoči *slow start* fazo. Ima tudi fast retransmit.
- TCP Vegas: izogibanje zamašitvam – ko se RTT poveča, se zmanjša **`CongWin`**.

Povzetek nadzora zamašitev

- **`CongWin`** < prag: slow start (eksponentna rast)
- **`CongWin`** > prag: congestion avoidance (linearna rast)
- 3 duplikati ACK:
  prag = **`CongWin`**/2, **`CongWin`** = prag.
- Timeout: Prag = **`CongWin`**/2, **`CongWin`** = 1

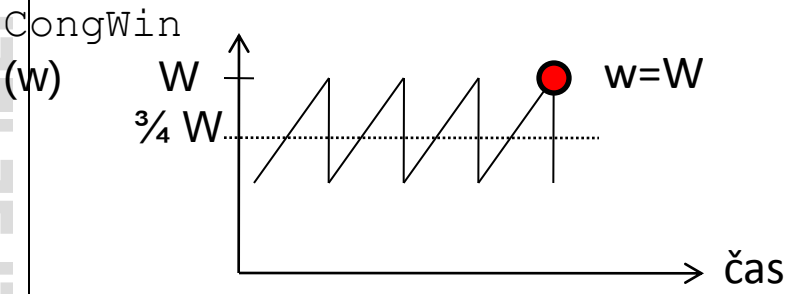
Primer: TCP Reno



Kdaj je počasen začetek in kdaj izogibanje zamašitvam?
Kdaj so bili 3 duplikati ACK in kdaj timeout?
Kakšen je bil prag na začetku, kakšen ob T=18, in T=24?
Kdaj je bil poslan 70. segment?
Če bi pri T=26 imeli 3 duplikate ACK, kako bi določili prag in CongWin?

Kakšen je povprečen pretok?

- Pretok = CongWin/RTT



- Predpostavimo: W in RTT sta ~ konstantna.
  Potem pretok niha med W/(2 RTT) in W/RTT.
- Povprečen pretok = ¾ W/RTT

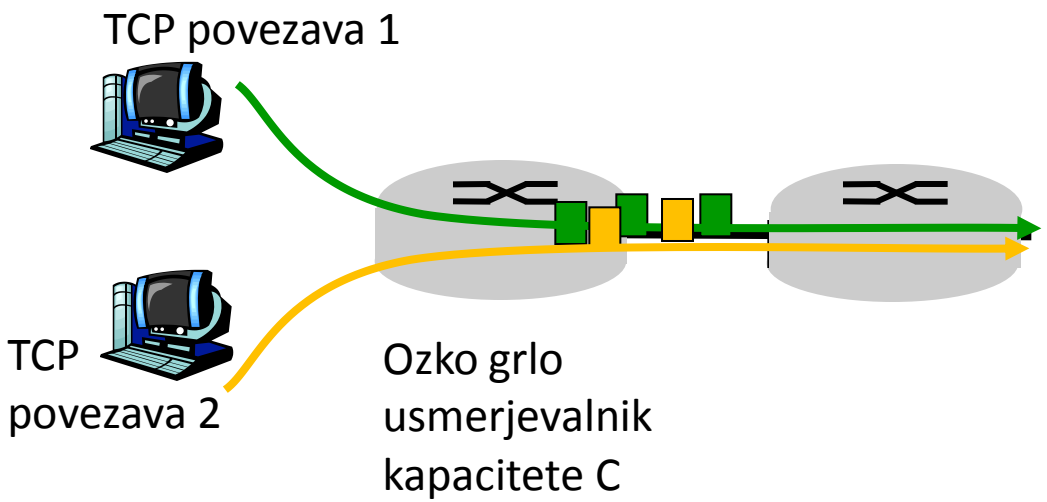
Primer

- Max. segment (MSS) = 1500 bytov
- RTT = 100 ms
- Če želimo 10 Gb/s pretoka: **`CongWin`** mora biti povprečno 83.333 !

- Možno je izračunati, da za ta pretok lahko izgubimo le $2*10^{-10}$ segmentov (enega na 5 milijard poslanih...) !

Je TCP pravičen?

- Cilj: Vsaka od N TCP sej po isti povezavi s kapaciteto C naj bi dobila C/N .



Pravičen ?

- TCP sam: DA
- Več paralelnih TCP povezav za isto aplikacijo in ena povezava za drugo aplikacijo: z vidika aplikacije NI pravičen
- UDP in TCP po istem omrežju: ni pravično do TCP (UDP pošilja brez omejitev pretoka)

Sklep

- Principi za transportnimi storitvami
  - (de)multipleksiranje
  - Zanesljiv prenos
  - Kontrola pretoka in zamašitev
- TCP in UDP