

Računalniške komunikacije

2023/24

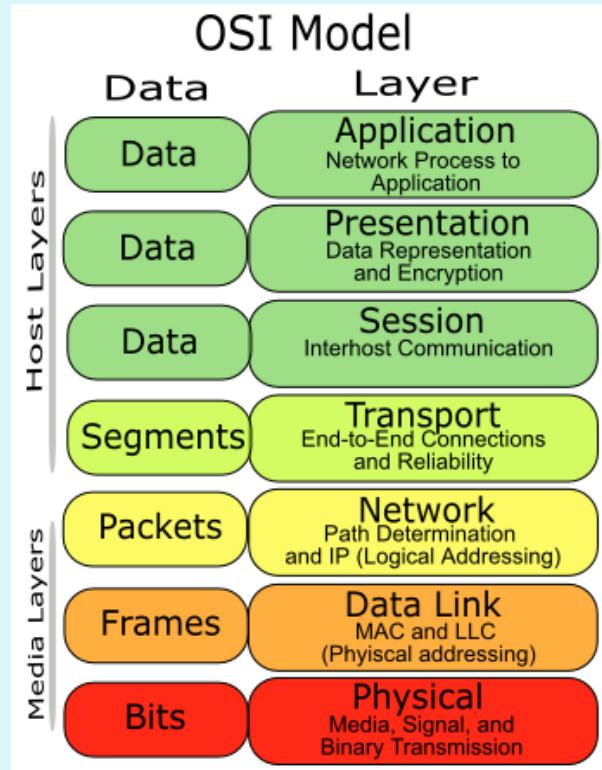
transportna plast
storitve, multipleksiranje,
protokol UDP,
konstrukcija TCP, potrjevanje,
protocol TCP,
nadzor pretoka, nadzor zasičenja
pravičnost TCP in UDP

Povzetek: omrežna plast

- **omrežna plast**

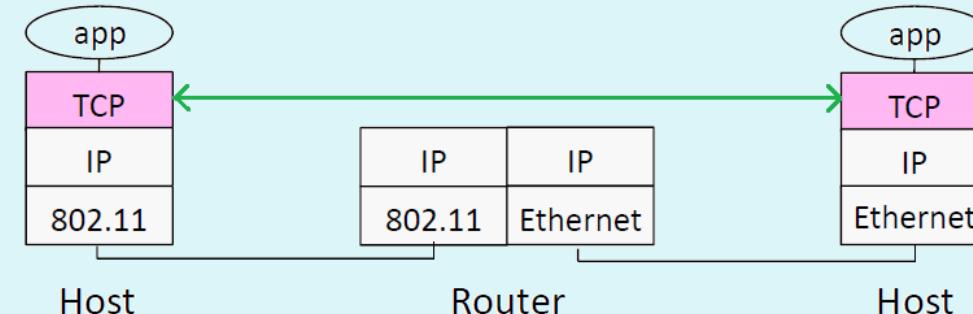
- funkcije usmerjevalnikov: **posredovanje in usmerjanje**
- **storitve** omrežne plasti (zagotovljena dostava, čas, zaporedje, pasovna širina, varianca zakasnitve, varnost)
- **modeli storitev** (best effort, CBR, ABR) in vrste omrežij (povezavna, nepovezavna)
- **protokol IPv4:**
 - oblika paketa, naslavljanje (32-bitni naslovi, podomrežja, hierarhija naslovnega prostora, razredi/CIDR)
 - fragmentacija, napadi z uporabo fragmentacije
 - dinamično dodeljevanje naslovov (DHCP), prevajanje omrežnih naslovov (NAT)
- **ICMP:** struktura datagrama, ping, traceroute
- **protokol IPv6**
 - potrebe: večji naslovni prostor (128-bitni naslovi), hitrejše usmerjanje, zagotavljanje kakovosti storitev
 - prehodni mehanizmi: dvojni sklad, tuneliranje
- **usmerjanje**
 - **centralizirani** algoritmi (gradijo drevo najkrajših poti)
 - **decentralizirani** (porazdeljeni) algoritmi (usmerjanje z vektorji razdalj):
 - iterativno računanje posredovalnih tabel
 - principa: good news travel fast, bad news travel slow
 - avtonomni sistemi (AS), intra-AS usmerjanje, inter-AS usmerjanje

Transportna plast



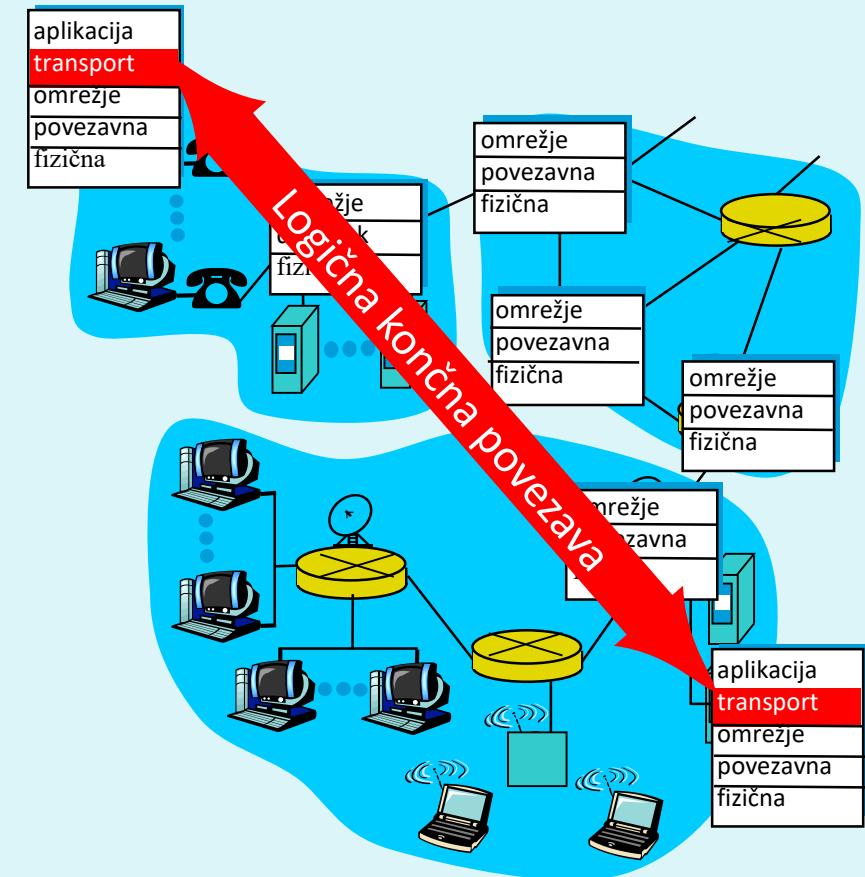
Naloge transportne plasti:

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



Storitve transportne plasti

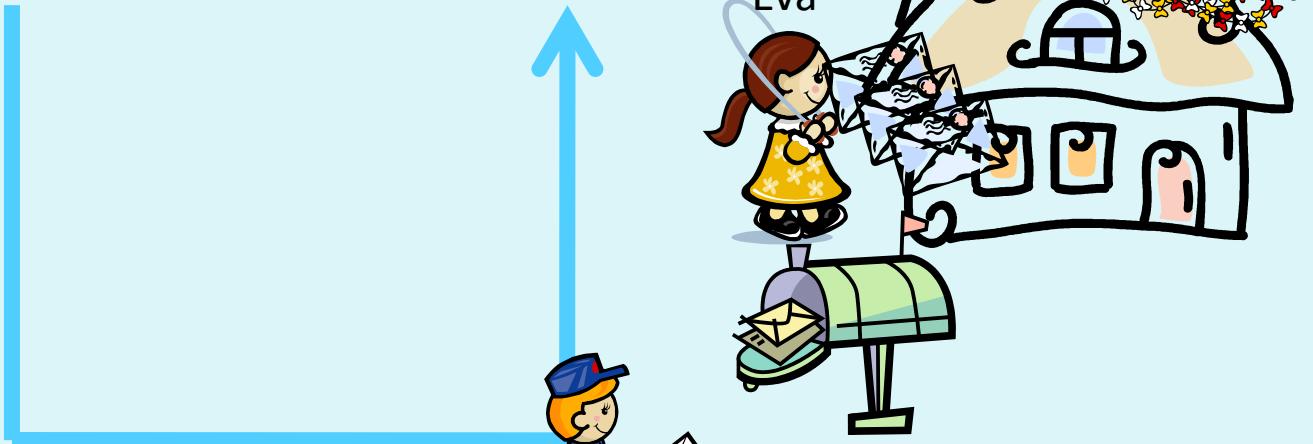
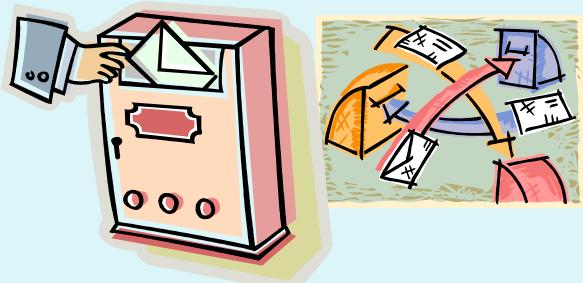
- Logična komunikacija med aplikacijskimi procesi
 - **pošiljatelj**: sporočilo razbije v SEGMENTE in jih posreduje v enkapsulacijo omrežni plasti
 - **prejemnik**: dekapsulira segmente iz paketov, sestavljene segmente združi v sporočila in jih posreduje aplikacijski plasti
- Protokola TCP in UDP



Analogija



Ana



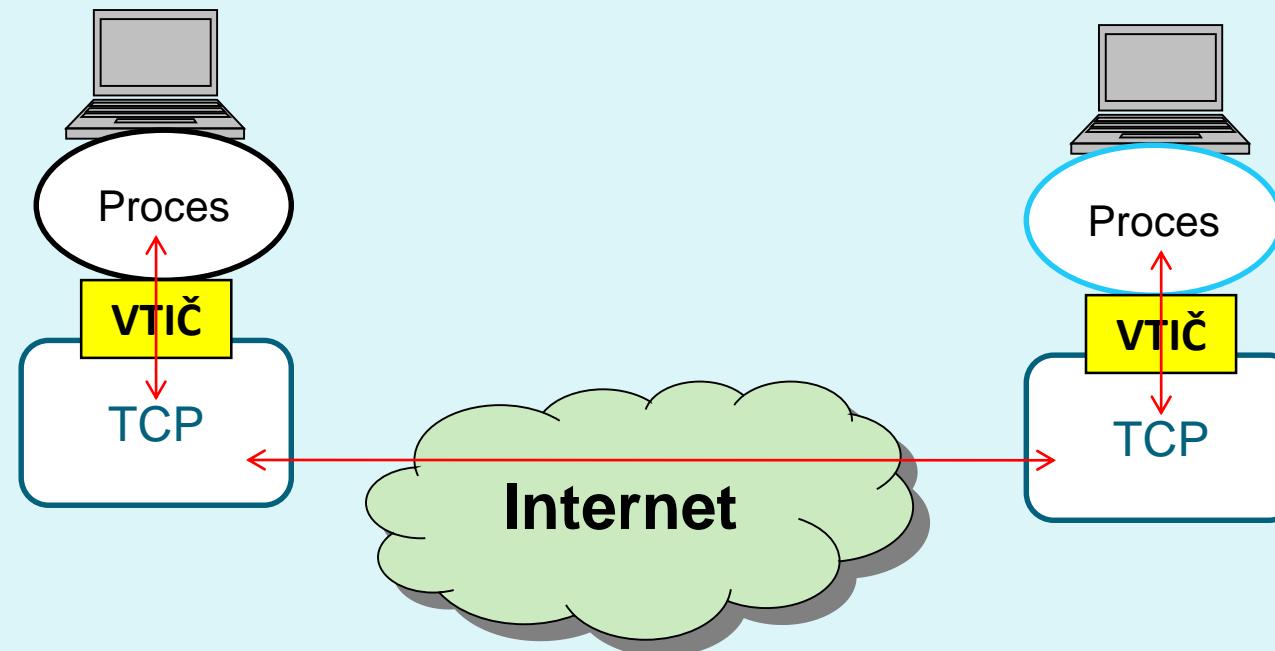
končni sistem = hiša
proces = otrok
aplikacijsko sporočilo = pismo
transportni protokol = Ana in Eva
omrežna plast = storitev pošte

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

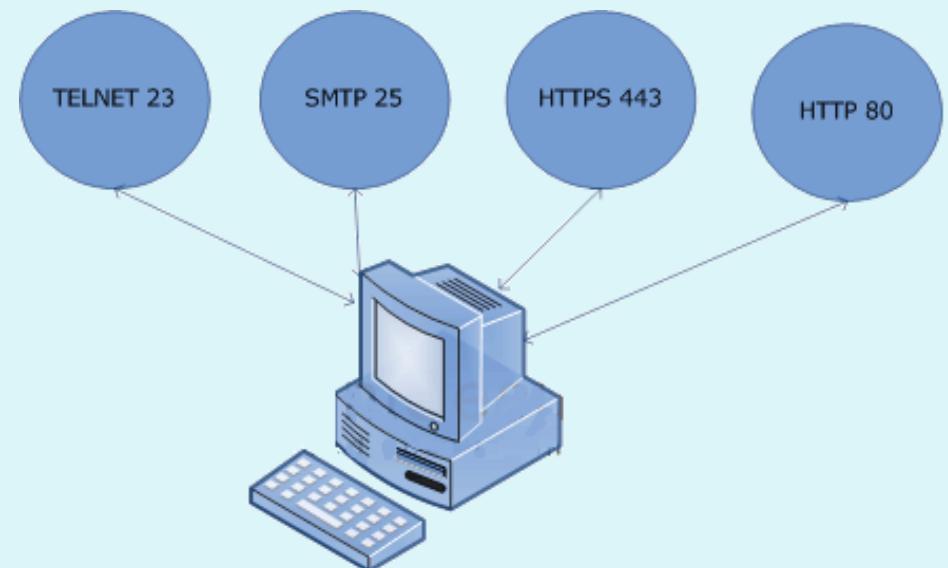
Kako komunicirati s procesom (aplikacijo)?

- 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 nasloviti proces na drugi strani?

- za naslavljjanje 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



Kako poteka demultiplexiranje?

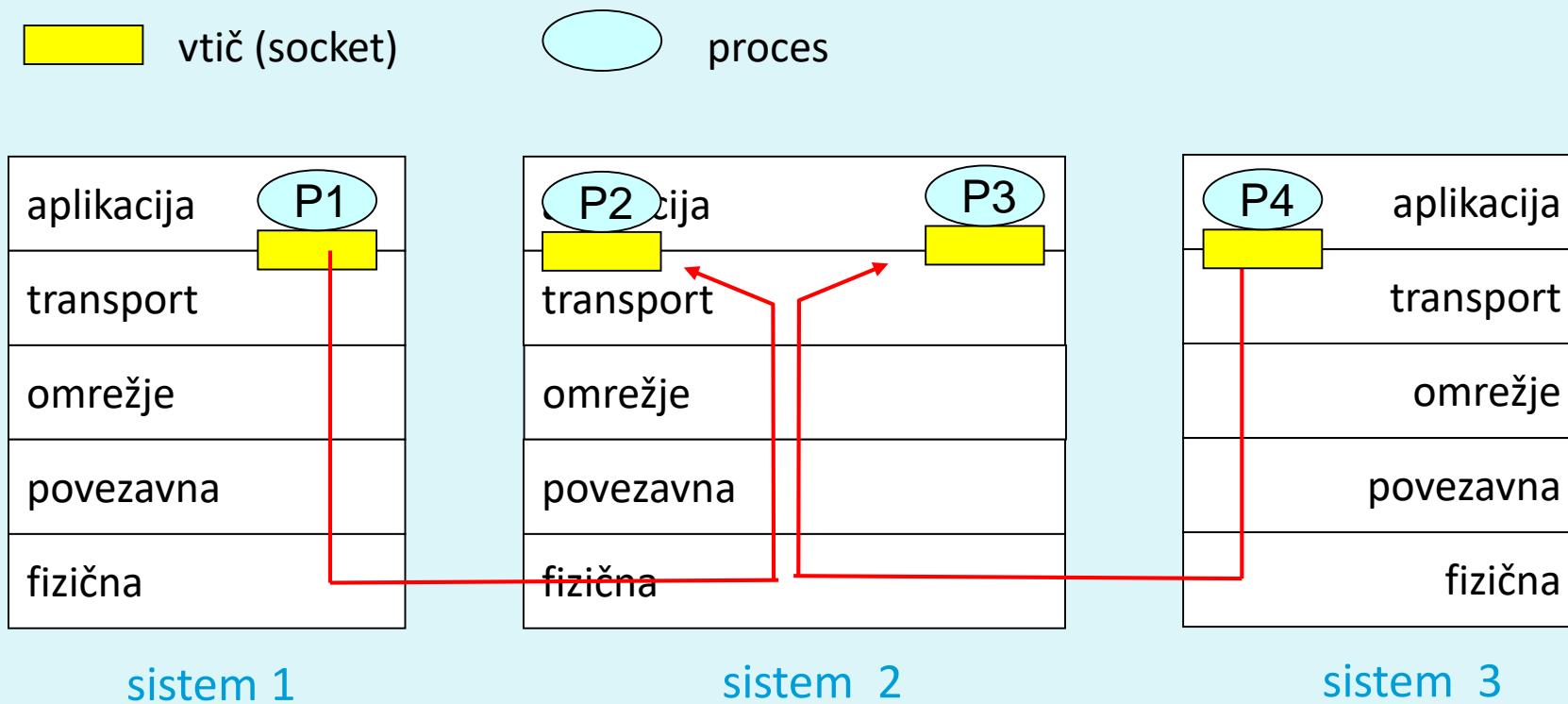
- vsak transportni segment potuje znotraj svojega paketa IP
- **transportni segment ima 16-bitna podatka:** številka vrat *izvora* (oznaka procesa, ki pošilja) in *ponora* (oznaka procesa na ciljni strani)



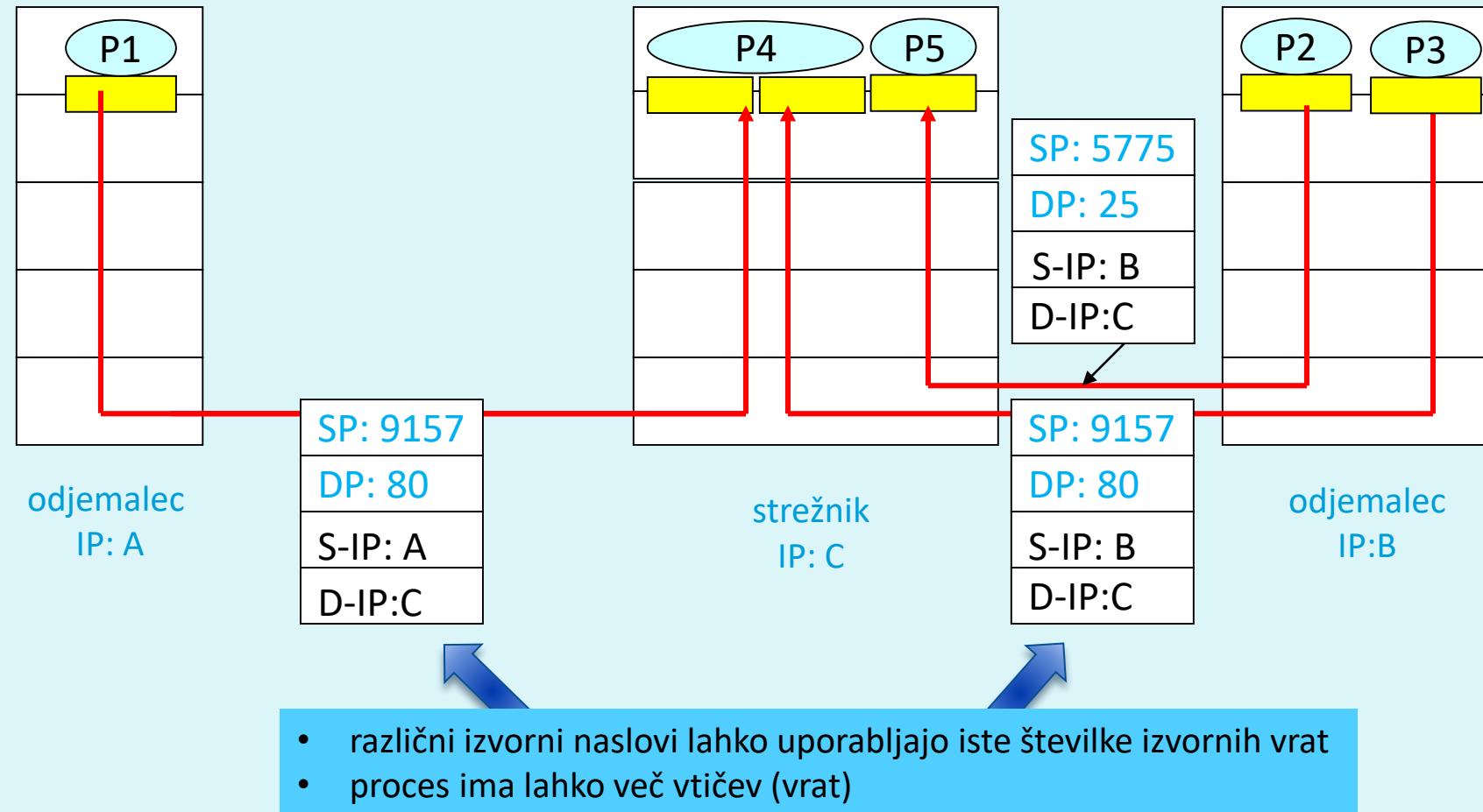
oblika segmenta TCP/
datagrama UDP

Multipleksiranje in demultipleksiranje

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



Povezavno demultiplexiranje (TCP)

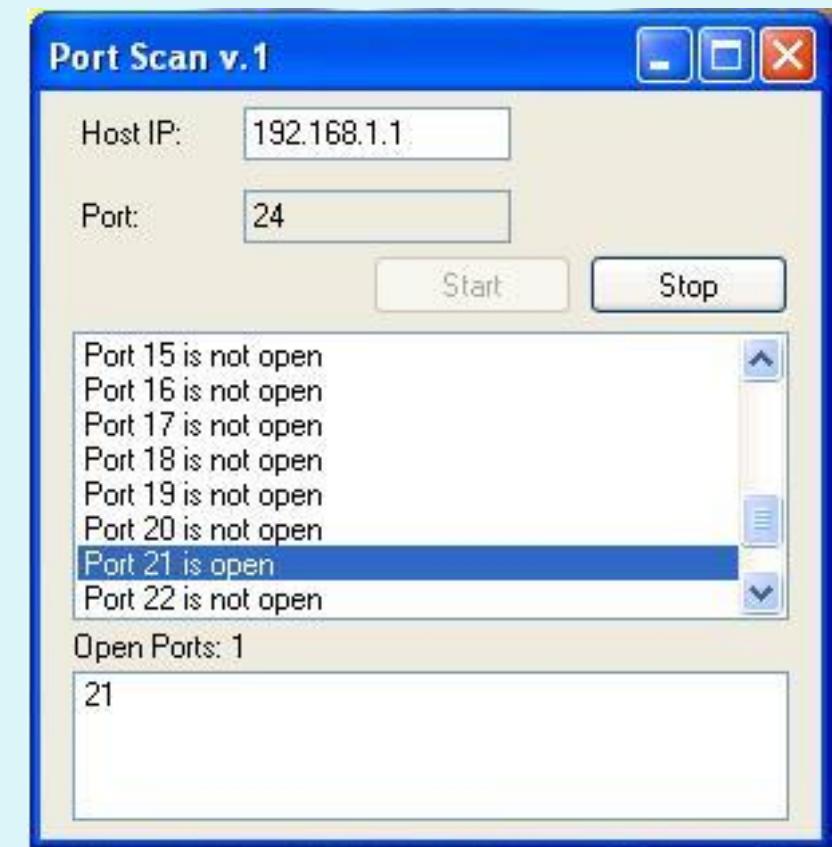


SP = source port, izvorna vrata (predstavlja naslov za odgovor)

DP = destination port, ciljna vrata (za naslavljjanje vtiča ciljnega procesa)

Varnost: Napad portscan

Napad **portscan** (pregled vrat) je namenjen pregledovanju strežnika, na katera vrata se bo odzival. S tem napadalec (ali administrator) dobi vpogled v procese, ki tečejo na strežniku. S poznavanjem šibkih točk strežniške programske opreme (npr. operacijski sistem, SQL server) lahko napadalec ogrozi delovanje sistema.

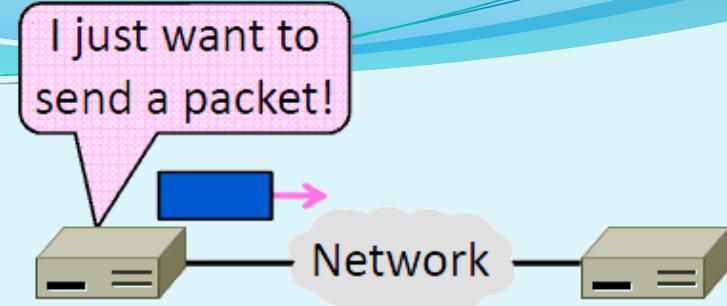


Nepovezavni transport: UDP

(User Datagram Protocol)



Storitve protokola UDP



LASTNOSTI

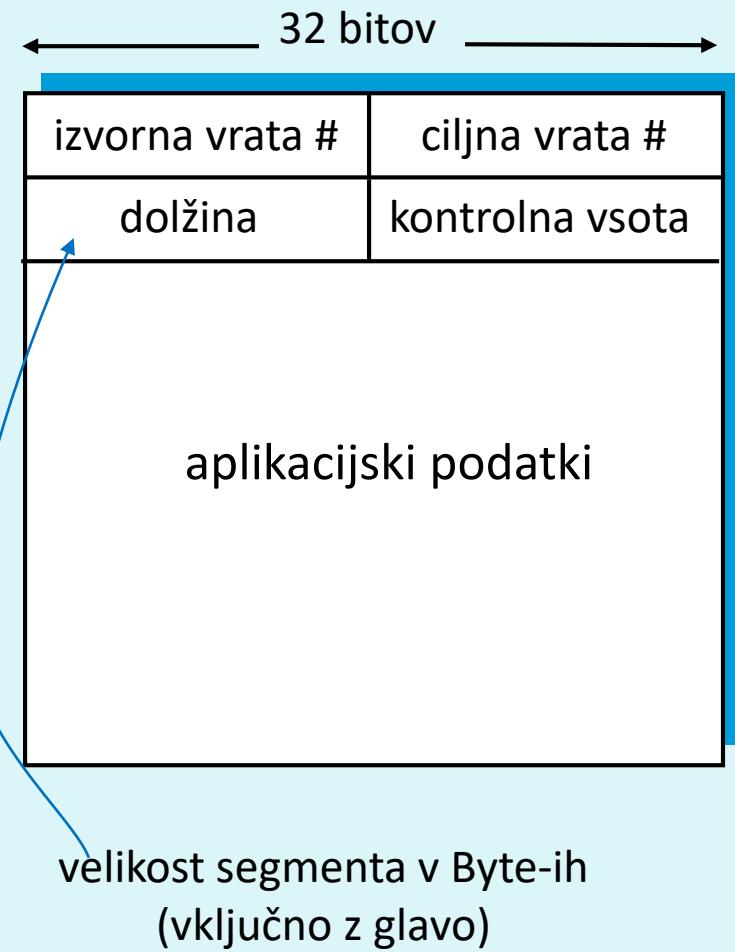
- nudi le "best-effort" storitev:
 - izgubljeni datagrami ("segmenti")
 - ne zagotavlja vrstnega reda
- nepovezaven (nima rokovanja)
- nima nadzora zamašitev

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

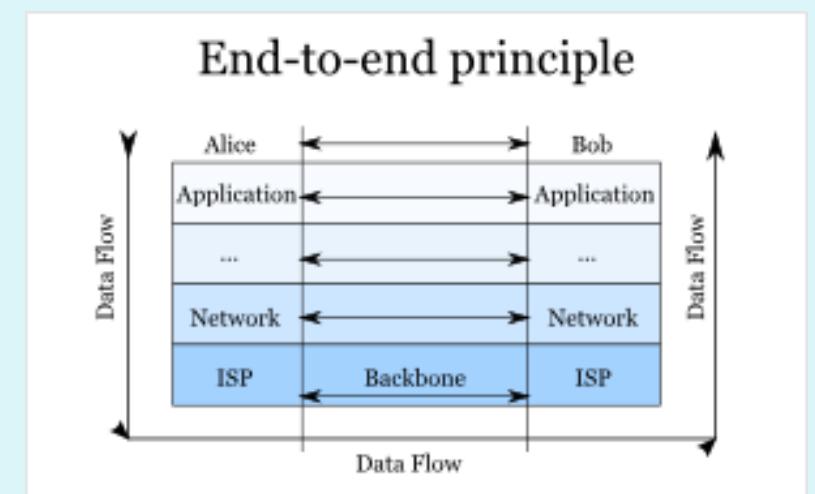
		$\begin{array}{r} 1110011001100110 \\ 1101010101010101 \\ \hline 110111011101110111 \end{array}$	
	prenos (se prišteje k rezultatu)	$\begin{array}{r} 110111011101110111 \\ \hline \end{array}$	
vsota		$\begin{array}{r} 1011101110111100 \\ 0100010001000011 \\ \hline 111111111111110 \end{array}$	
kontrolna vsota		$\begin{array}{r} 111111111111110 \\ \hline \end{array}$	

PREJEMNIK: izračuna vsoto
(prenos se prišteje k rezultatu)

PREJEMNIK: končna vsota

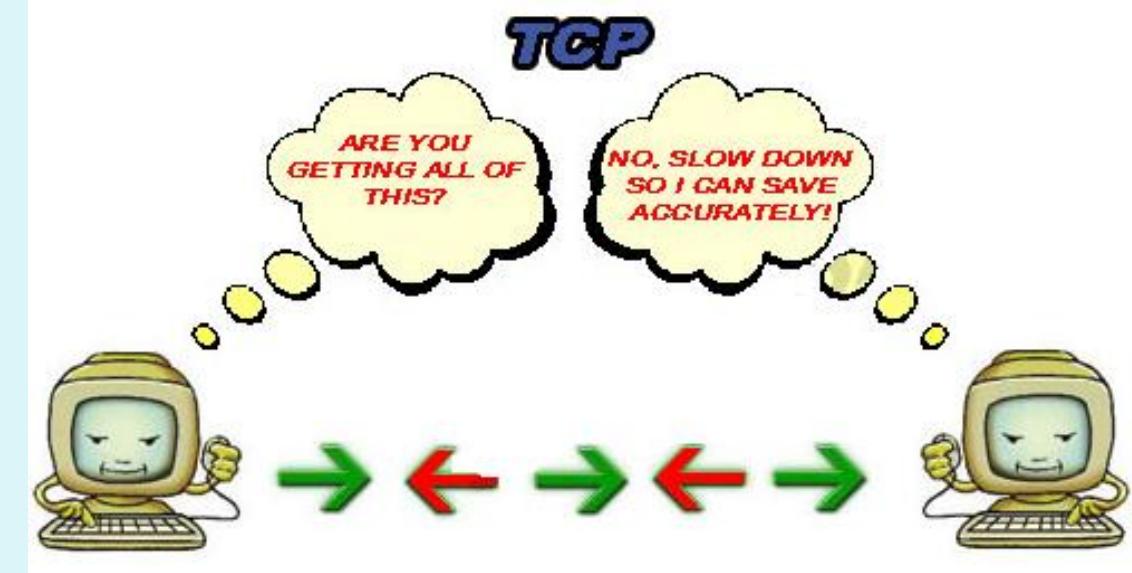
UDP: internetna kontrolna vsota

- zakaj datagram vsebuje kontrolno vsoto?
 - **ni zagotovila**, da protokoli na drugih plasteh zagotavljajo 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*)
 - nagradno vprašanje: kje smo dosedaj že omenili *princip končnih sistemov*?



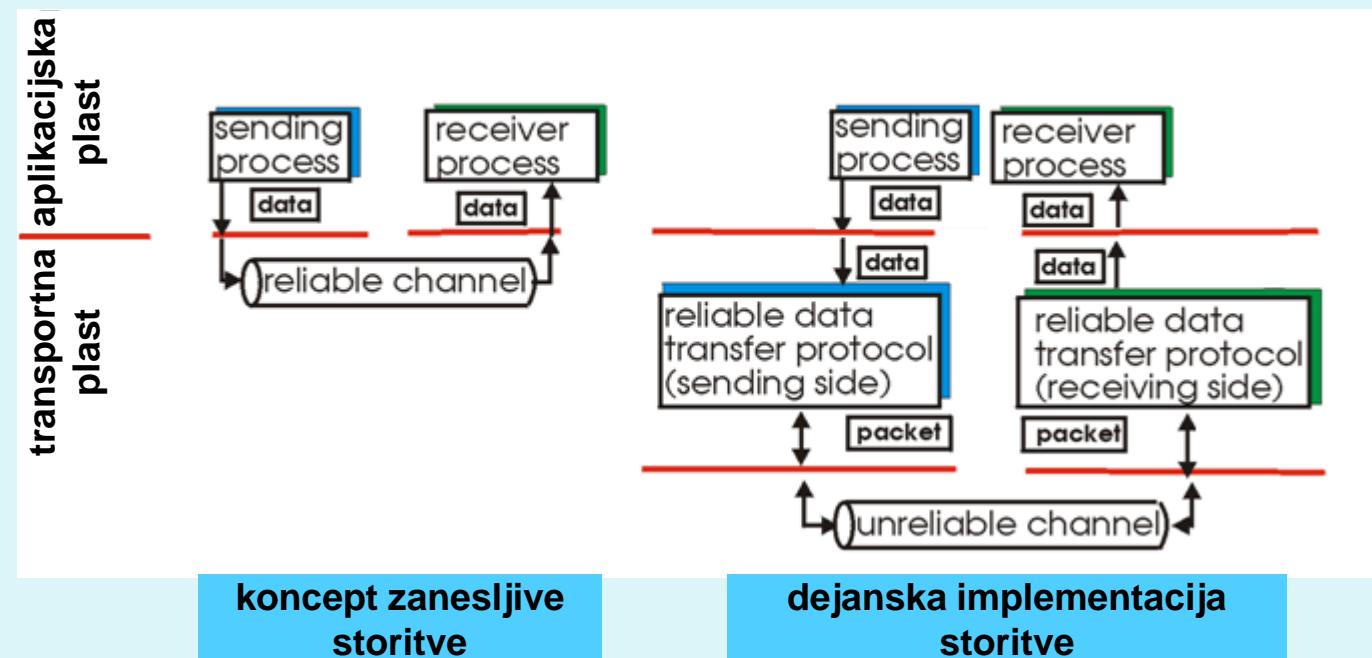
Povezavni transport: TCP

(Transfer Control Protocol)



Protokol TCP

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

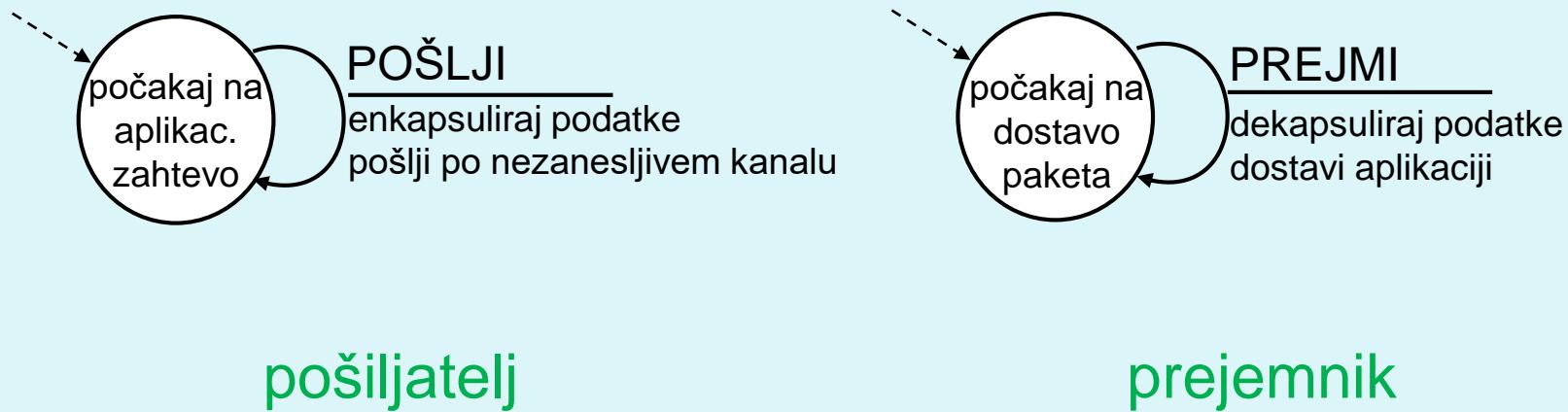


Protokol TCP

- konstruirajmo protokol, ki bo omogočal:
 - **osnovno funkcionalnost:** pošiljanje in prejemanje podatkov
 - **reševanje napak** pri prenosu (potrjevanje z ACK in NAK)
 - poenostavljeni potrjevanje (samo pozitivne potrditve ACK)
 - optimizirano potrjevanje (skupinsko potrjevanje paketov!)
 - **obravnavo izgubljenih** paketov (ponovno pošiljanje)
 - odpornost na izgubljene potrditve paketov

1. Enostavni protokol

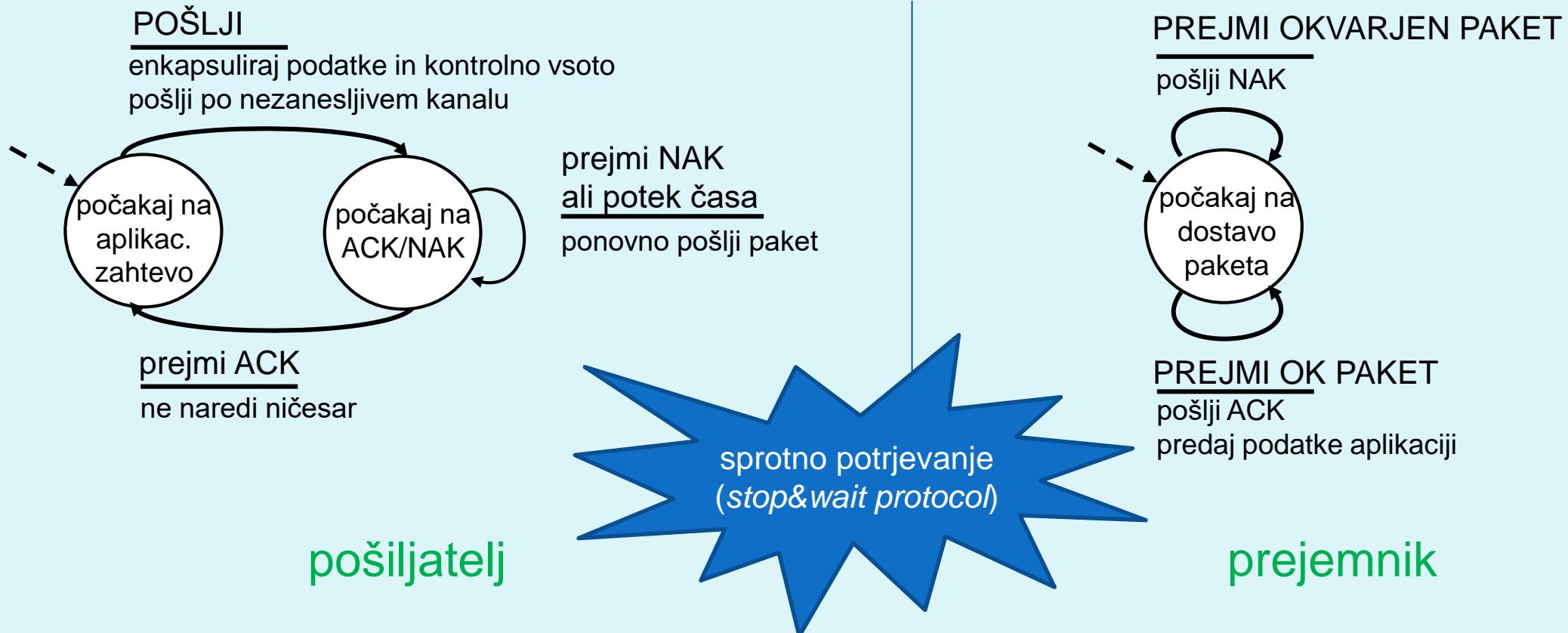
- ideja: vzemimo enostavni protokol in ga nadgradimo v zanesljivega
- delovanje protokolov pošiljatelja in prejemnika predstavimo s **končnim avtomatom** (vozlišče predstavlja stanje, povezava predstavlja prehod med stanji)



2. Reševanje iz napak pri prenosu

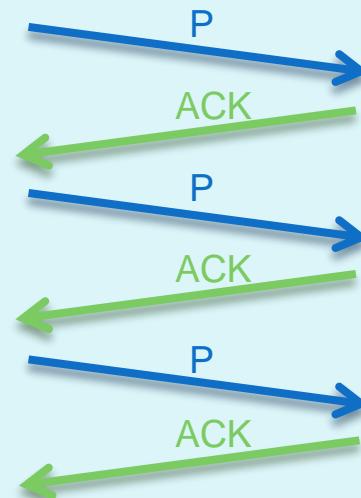
- vsebina paketov se lahko **okvari**, kar zaznamo s kontrolno vsoto
- dodamo funkcionalnosti:
 1. zaznavanje napak
 2. potrditve, ali je bil paket sprejet pravilno (ACK) ali nepravilno (NAK)
 3. čakanje na potrditev (časovni interval)
 4. ponovno pošiljanje ob napaki

2. Reševanje iz napak pri prenosu

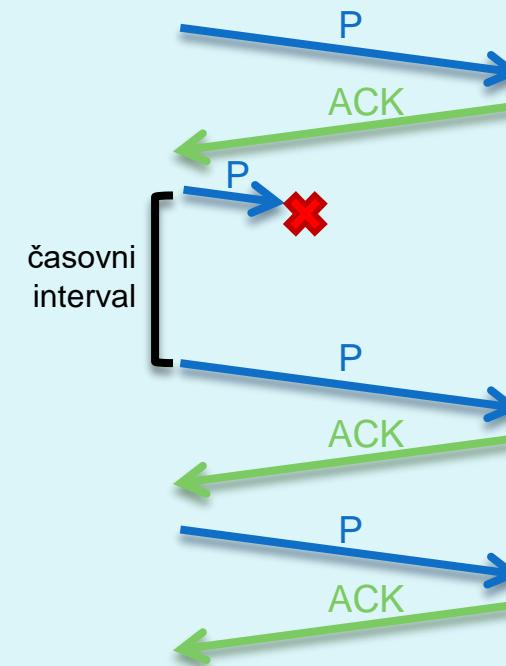


3. Izguba paketov ali potrditev

- novi problem: paketi se lahko **izgubijo** in ne pridejo do prejemnika, kar zaznamo s tem, da jih prejemnik ne potrdi
- REŠITEV: pošiljatelj **počaka določen interval časa** (potrebujemo štoparico!) na ACK in če ga ne prejme, pošlje paket ponovno



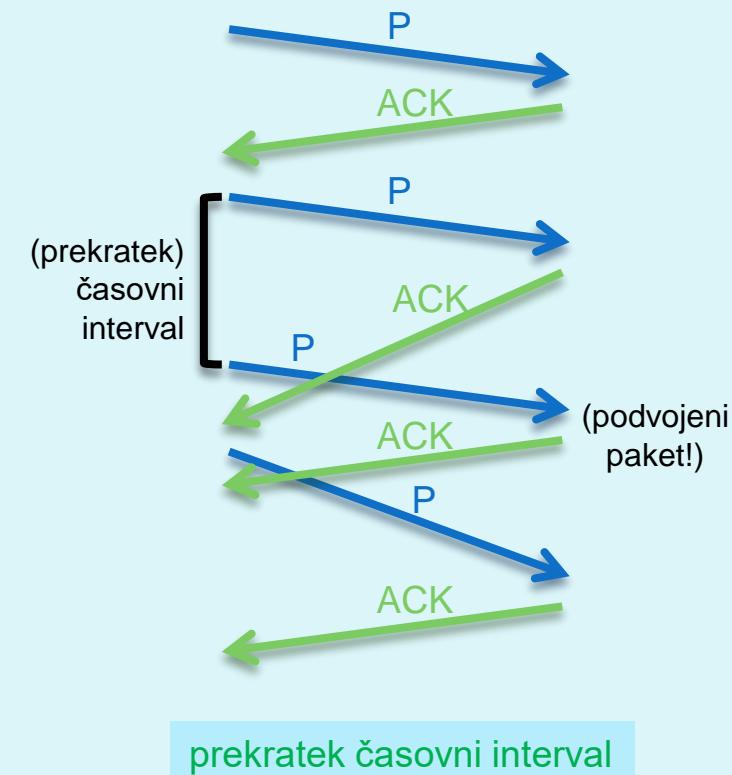
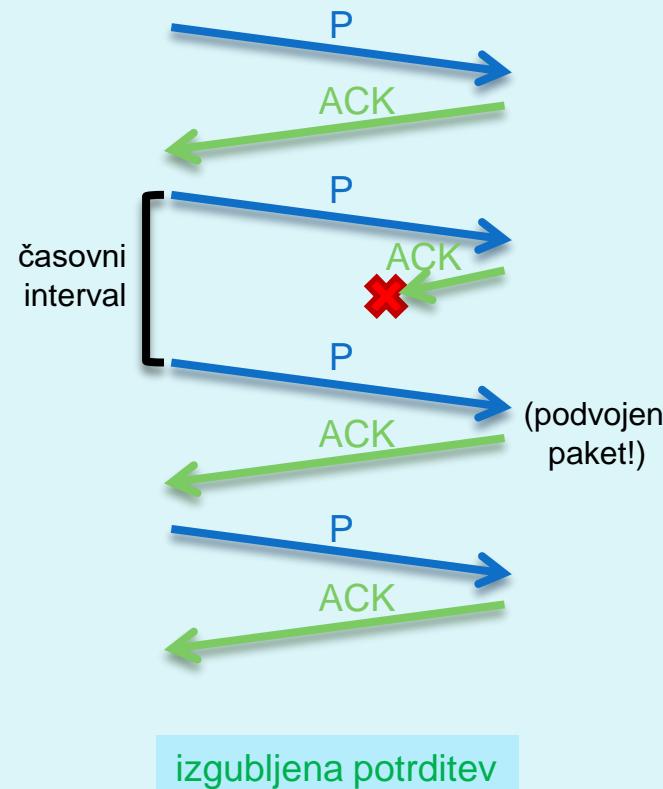
brez izgube paketa



z izgubo paketa

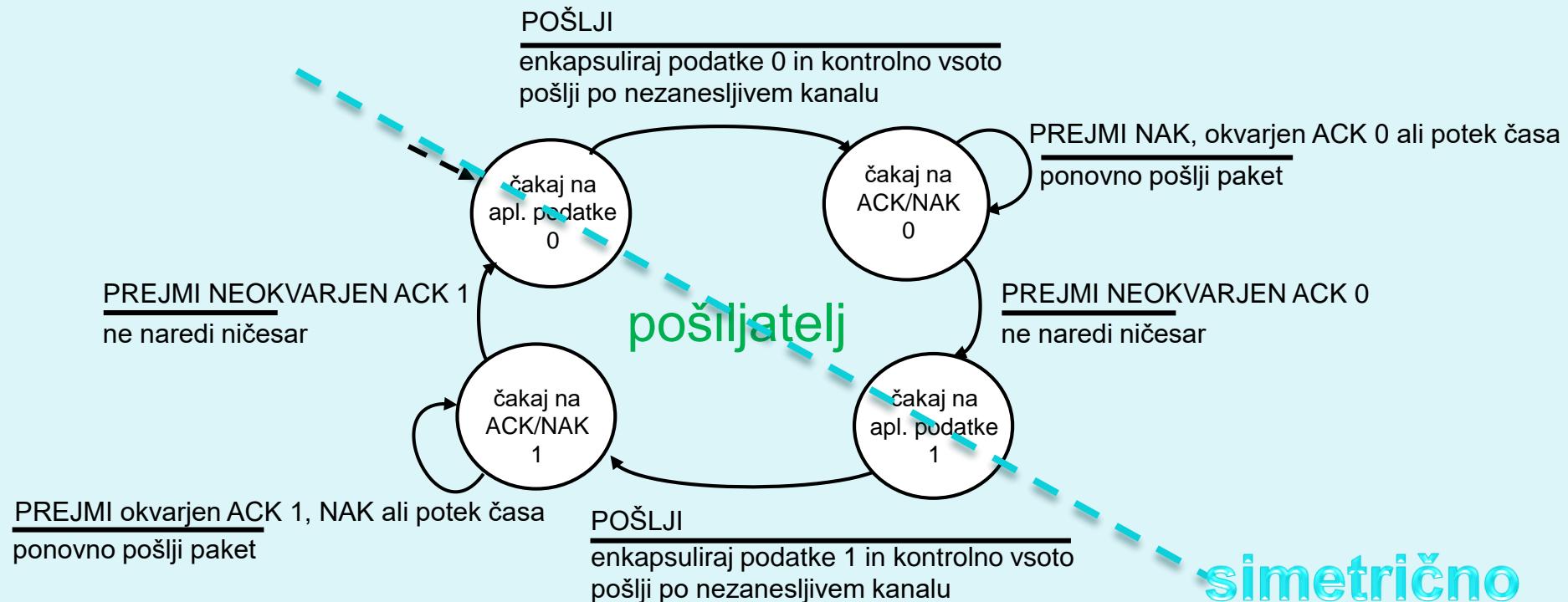
3. Izguba paketov ali potrditev

- možna je tudi **izguba potrditve** (povzroči prejetje podvojenega paketa)
- možnost **prekratkega časovnega intervala** (tudi tu pošiljatelj pošlje podvojeni paket)
- težavo rešimo s **številčenjem paketov** in zaznavanjem istih številk

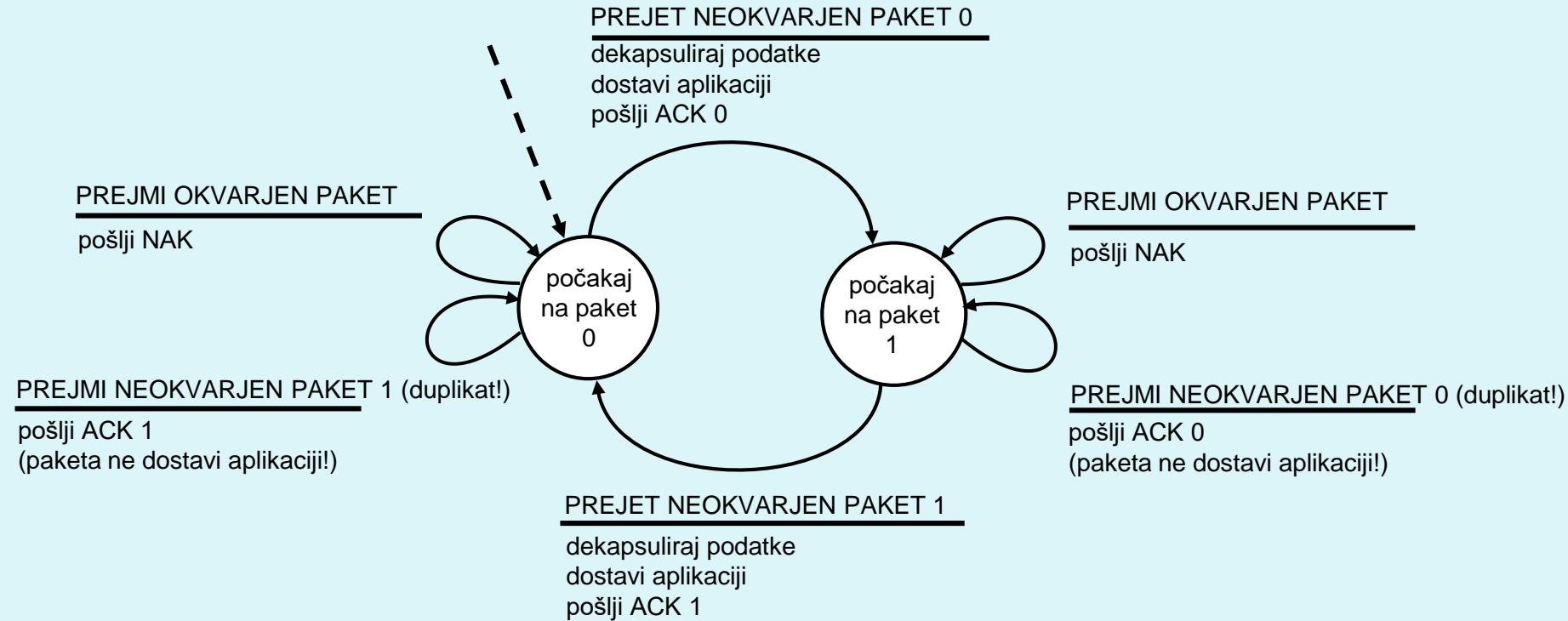


3. Izguba paketov ali potrditev

- REŠITEV: paketom dodamo zaporedne številke, s katerimi ločimo podvojene pakete



3. Izguba paketov ali potrditev



prejemnik

4. Izboljšava: posredno potrjevanje (samo ACK)

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

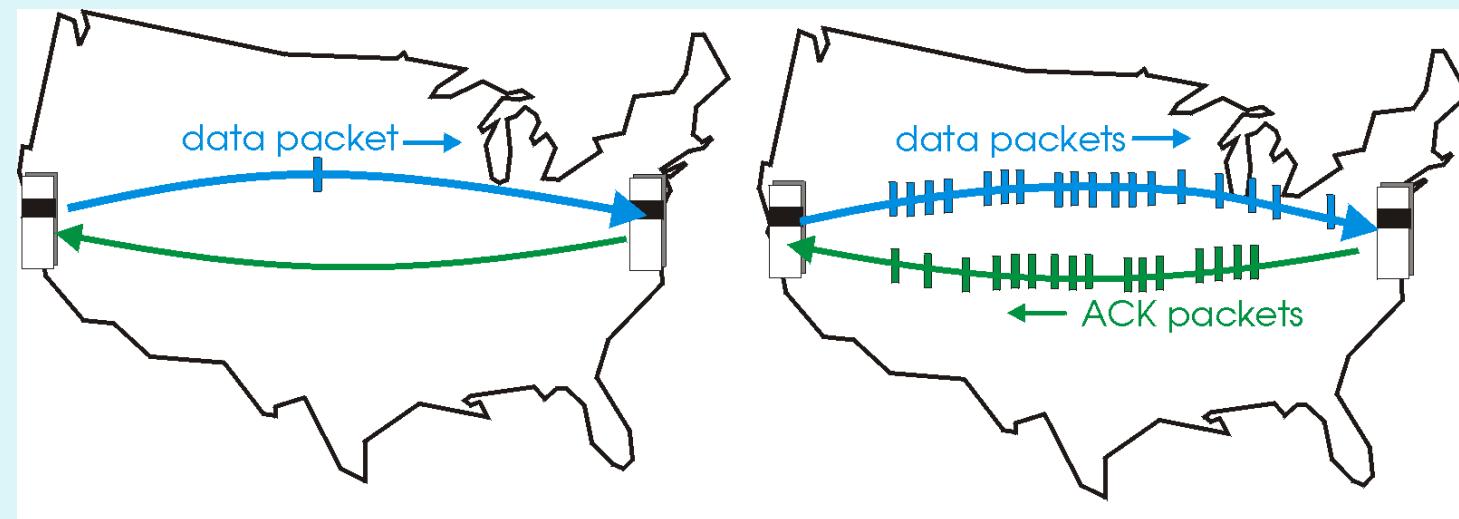
NEPOSREDNO POTRJEVANJE: uporaba ACK in NAK

POSREDNO POTRJEVANJE: uporaba samo ACK

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

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**



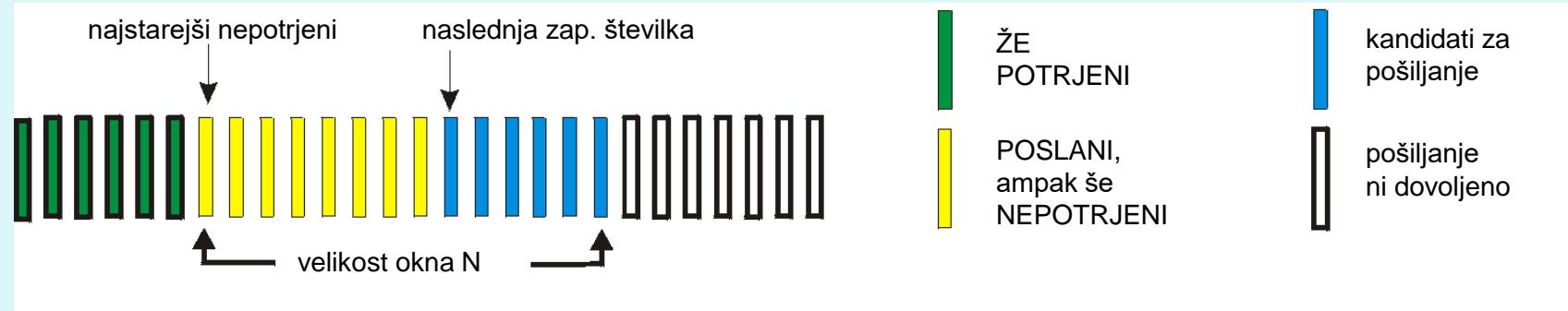
- ker lahko istočasno potuje več paketov, brez sprotnega čakanja na njihovo potrditev, je izkoriščenost kanala večja

5. Tekoče pošiljanje

- potrebujemo:
 - večji razpon števil za številčenje paketov
 - 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*)

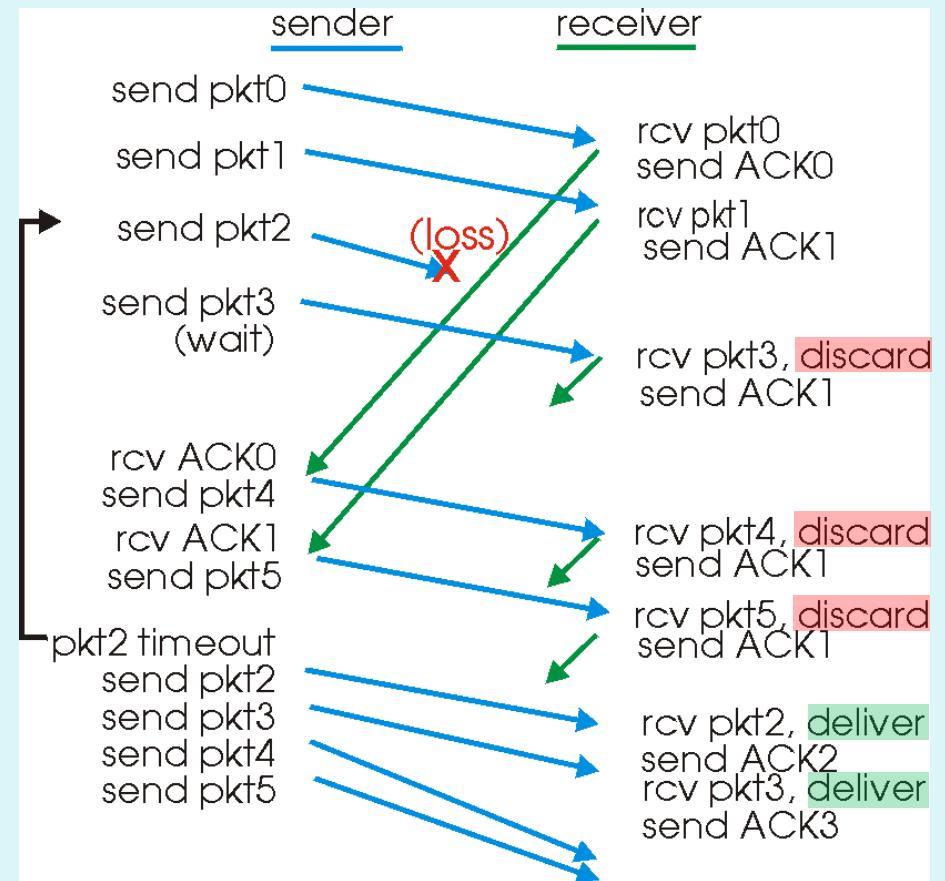
5. Tekoče pošiljanje: ponavljanje N nepotrjenih

- 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, pošlji vse nepotrjene pakete v oknu ponovno
- DEMO: [applet](#)



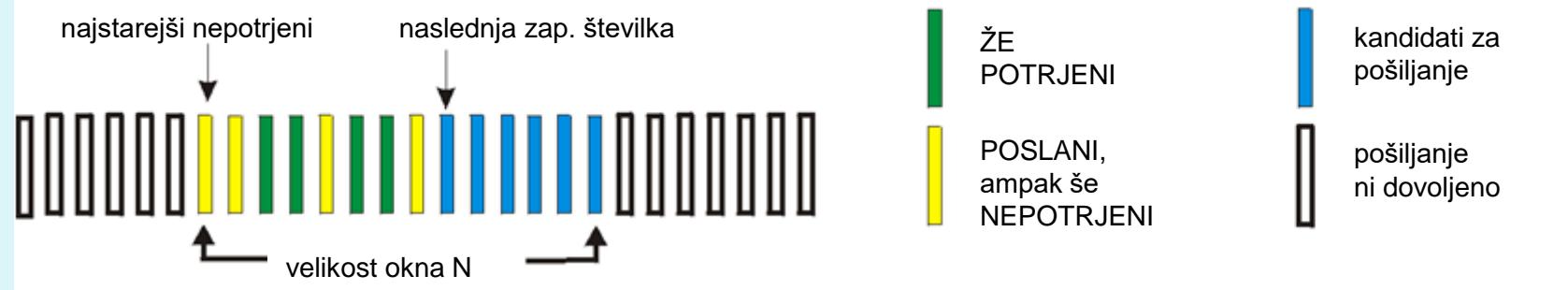
5. Tekoče pošiljanje: ponavljanje N nepotrjenih

- prejemnik lahko prejme **podvojene pakete**, ko pošiljatelj pošlje vse nepotrjene -> razpozna jih po zaporednih številkah in zavri
- prejemnik lahko prejme pakete v **napačnem vrstnem redu** -> te zavri, saj bodo poslani ponovno, ko poteče štoparica za najstarejšega

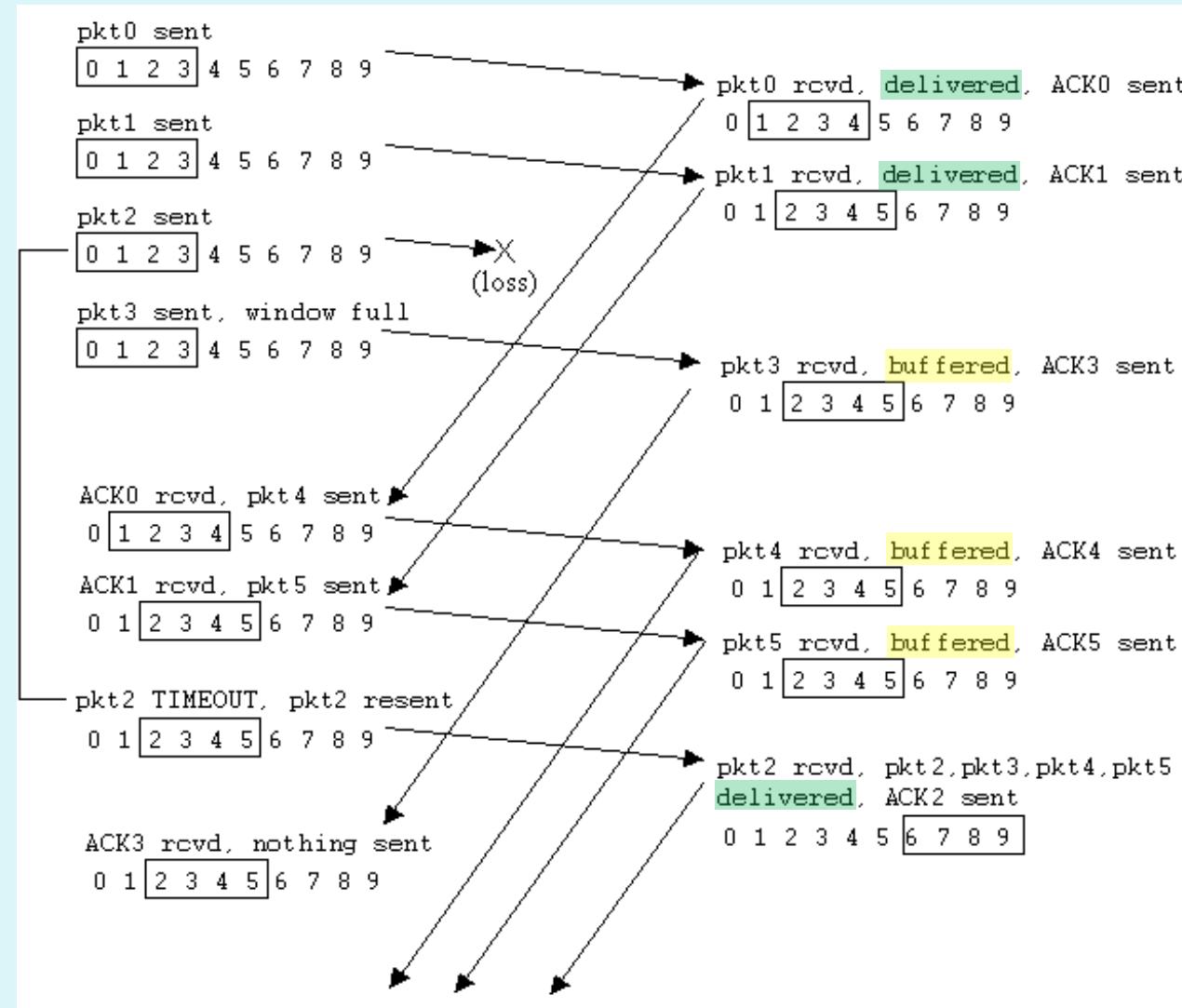


5. Tekoče pošiljanje: ponavljanje izbranih

- prejemnik potrujuje 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
- pošljatelj hrani **štoparico** za **vsak posamezni nepotrjeni** paket
- DEMO: [applet](#)



5. Tekoče pošiljanje: ponavljanje izbranih



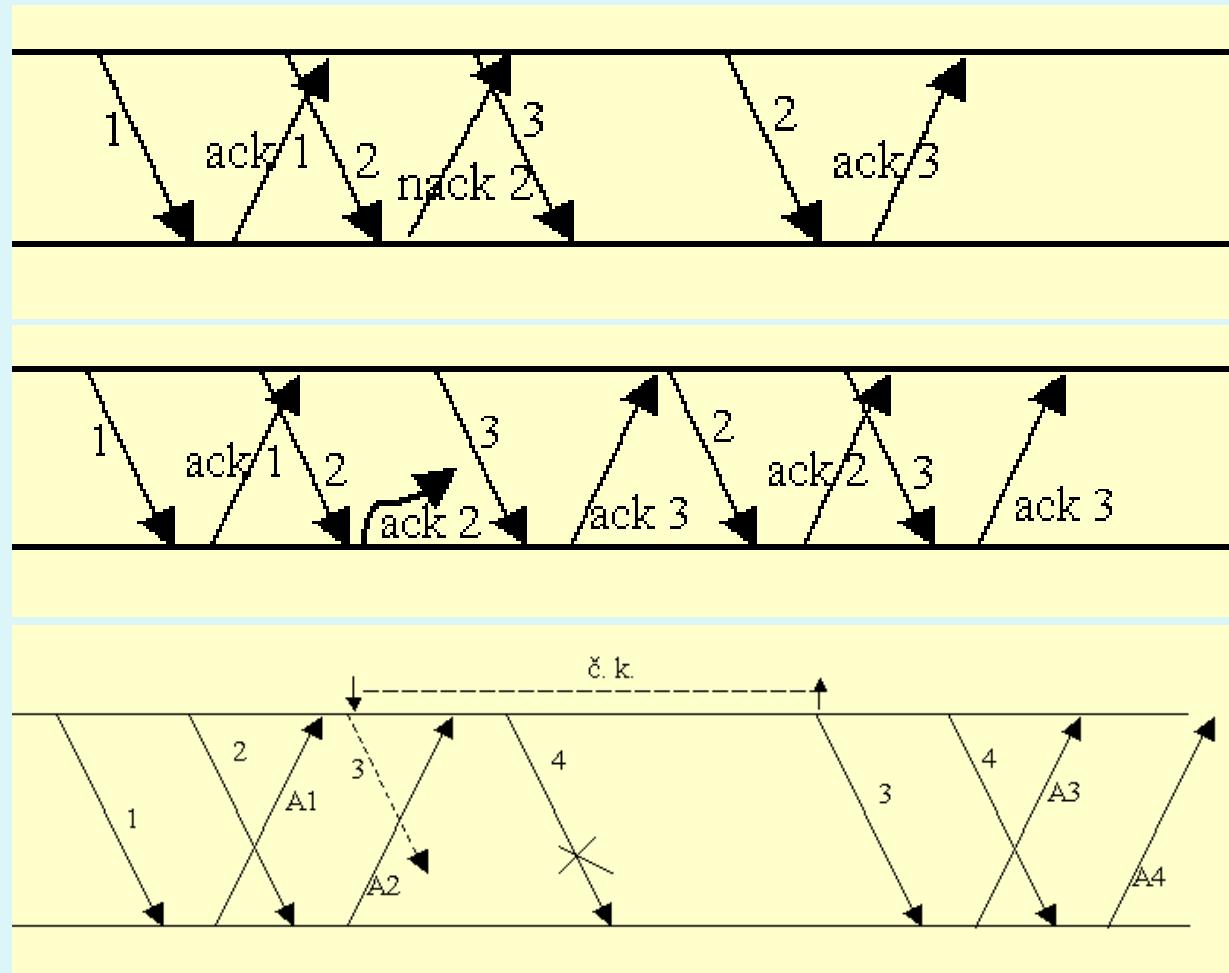
Potrjevanje

- 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

	SPROTNO	TEKOČE pošiljanje
NEPOSREDNO	✓	✓
POSREDNO	✓	✓

Vaja 1

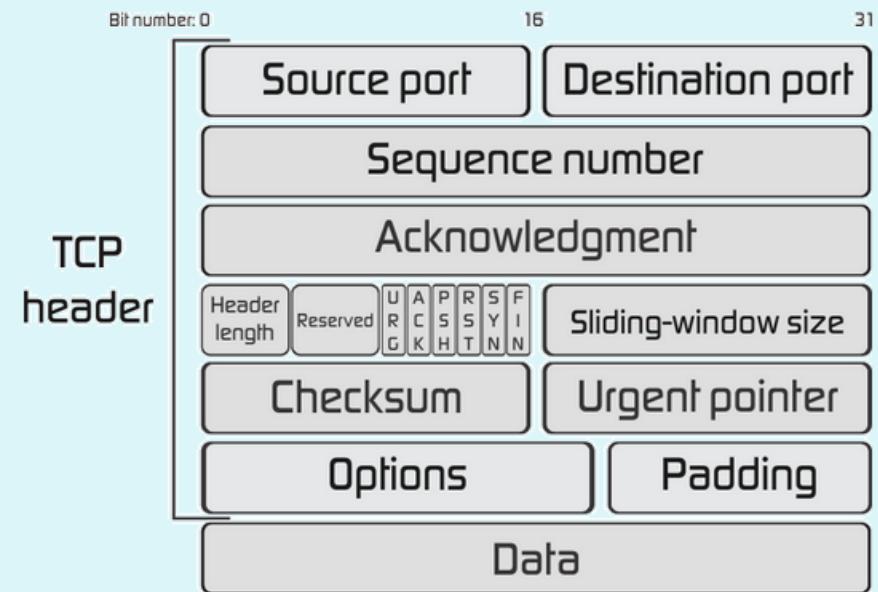
- Za katere tipe potrjevanj gre v spodnjih primerih?



Vaja 2

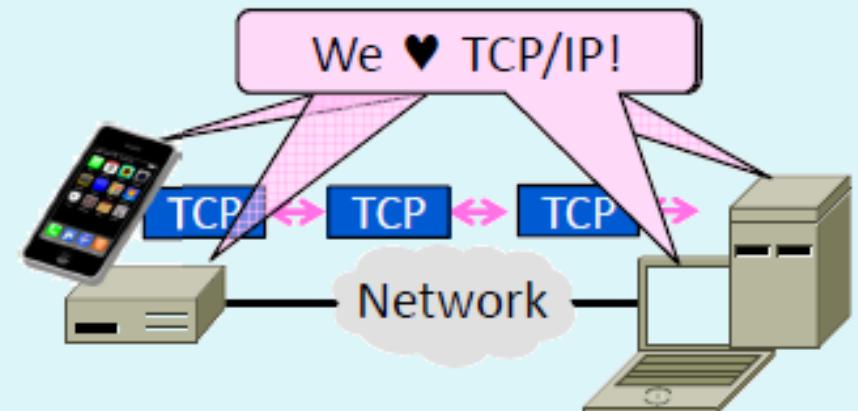
- Uporabljamo tekoče pošiljanje 6 paketov s protokolom za ponavljanje le izbranih paketov. Širina okna je 4.
- Nariši shemo prenosov, če se izgubita 1. in 3. paket, po ponovitvi pa še potrditev 1. paketa?

Protokol TCP

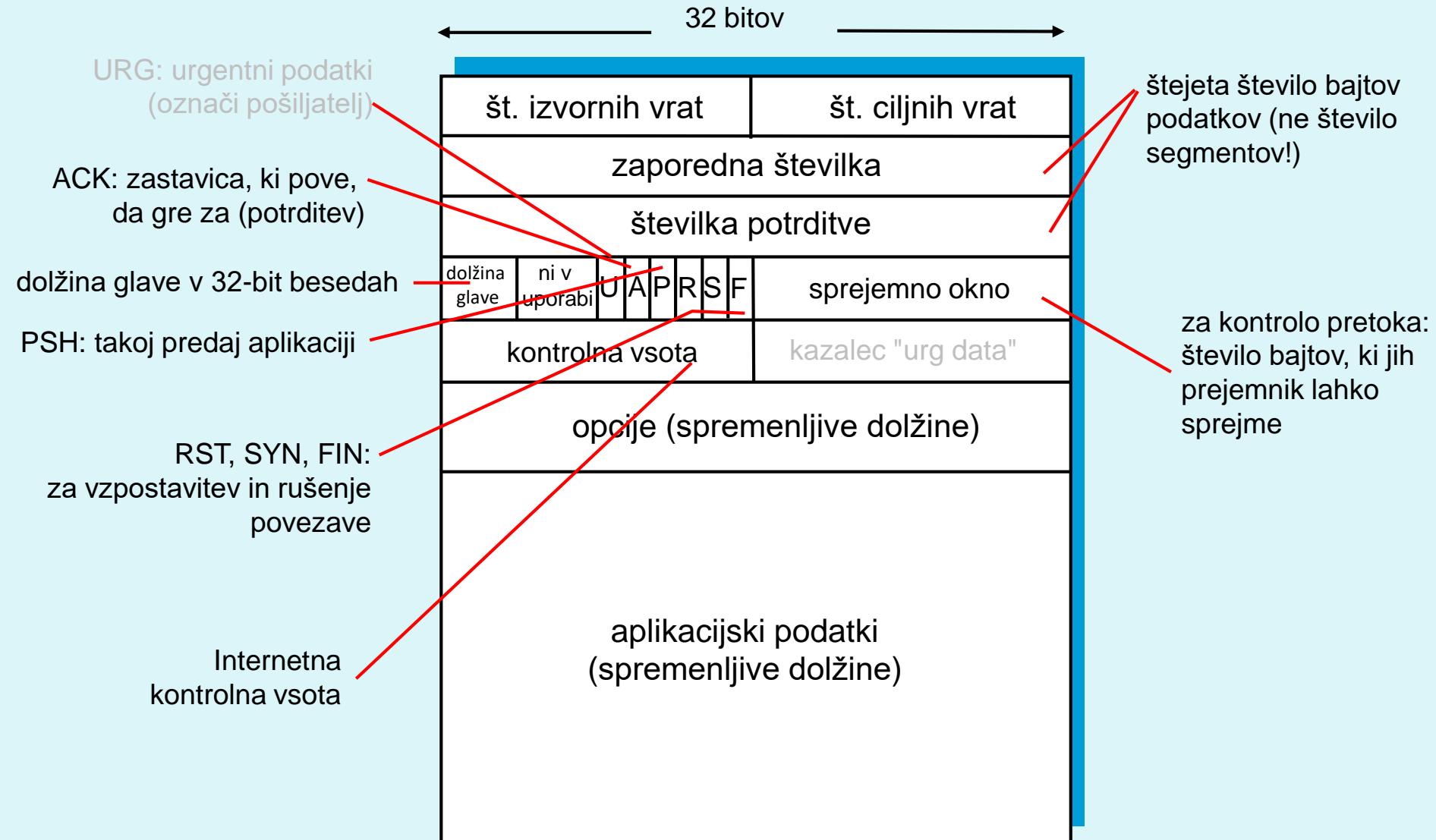


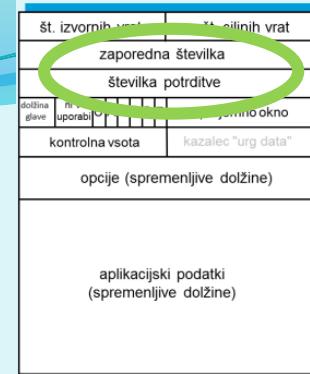
Lastnosti protokola TCP

- 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 **kontrolu pretoka** (angl. *flow control*, pošiljatelj ne preobremeniti prejemnika)
- ima **kontrolu zasičenja** (angl. *congestion control*, pošiljatelj ne preobremeniti omrežja)
- uporablja **tekoče pošiljanje**, velikost okna se avtomsatsko določa glede na kontrolu pretoka in kontrolu zasičenja



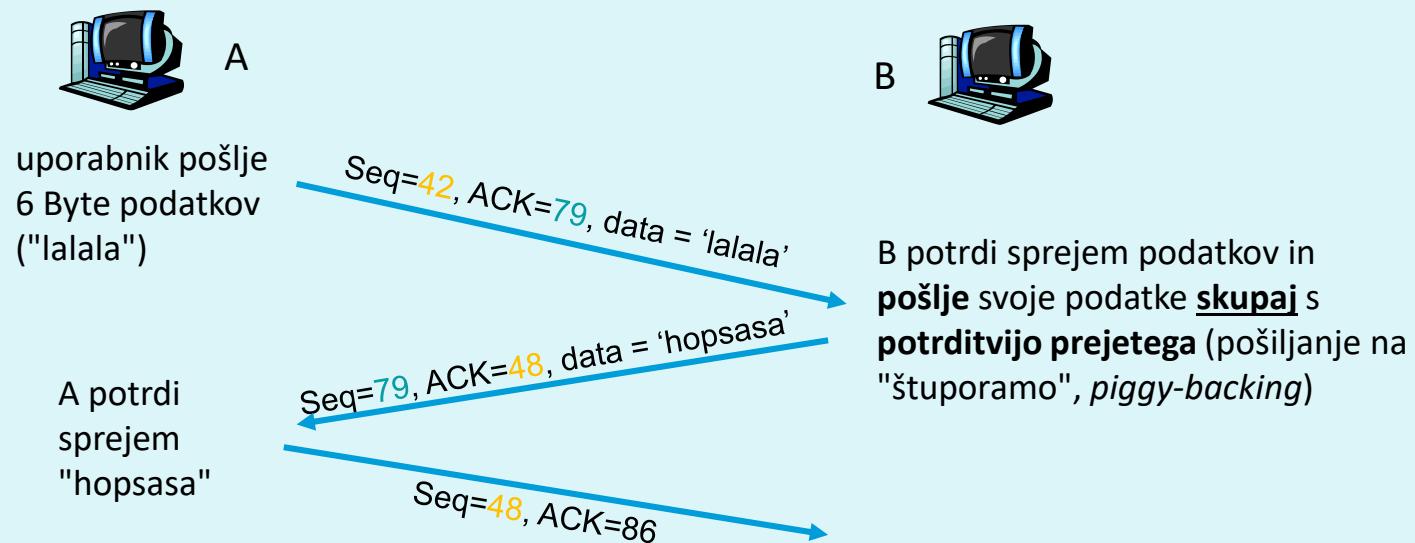
TCP segment

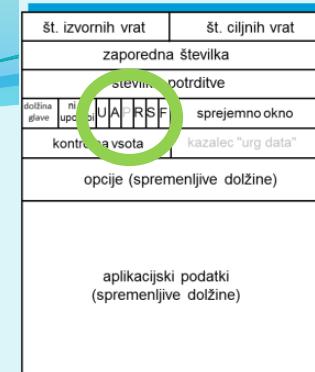




Številčenje segmentov in potrditev

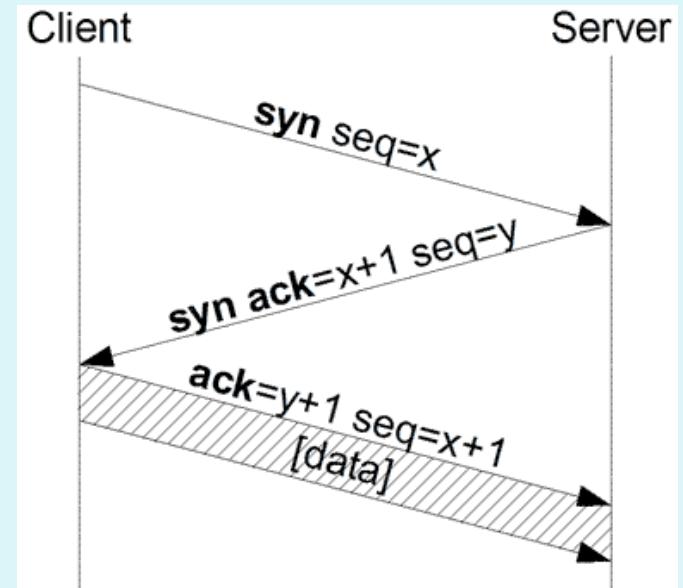
- pošiljatelj in prejemnik najprej VZPOSTAVITA ZVEZO. Povezava je nato **dvosmerna** (vsak lahko pošilja drugemu)
- pošiljatelj lahko v enem segmentu **istočasno** pošlje **nove podatke in potrditev (ACK)** prejšnjega segmenta
- številke pomenijo:**
 - SEQ (zaporedna številka):** številka prvega Byte-a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





TCP: vzpostavljanje povezave

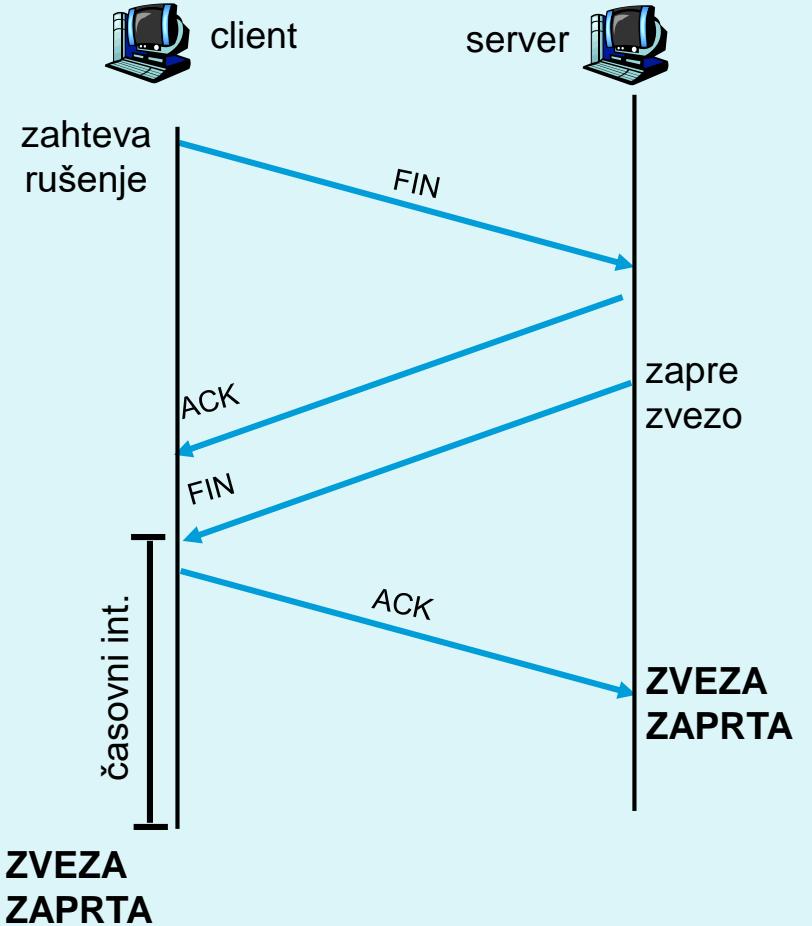
- pošiljatelj in prejemnik pred pošiljanjem izvedeta rokovanje (handshake), v katerem izmenjata parametre:
 - **začetne pričakovane zaporedne številke** (naključno določene)
 - **velikosti medpomnilnikov** (za kontrolo pretoka)
- trojno rokovanje (*three-way handshake*)
 1. odjemalec pošlje segment z 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 ("štuporama")



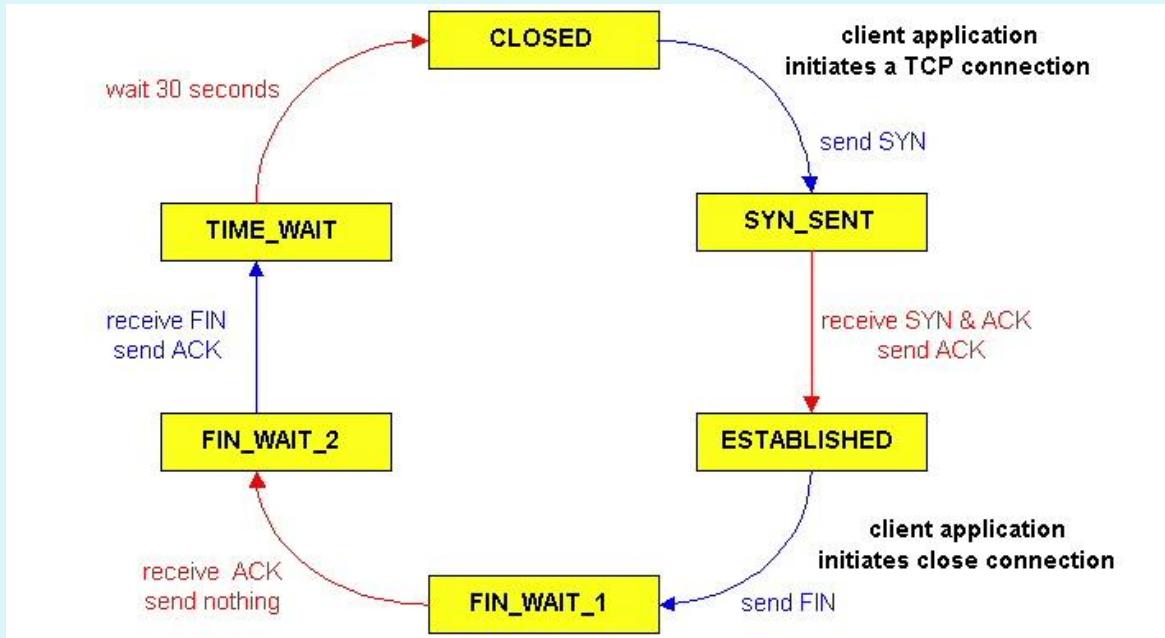
št. izvornih vrat	št. ciljnih vrat
zaporedna številka	
število potrditve	
dolžina glave	ni v uporabi
UA	R S F
sprejemno okno	kazalec "urg data"
kontrolna vsota	
opcije (spremenljive dolžine)	
aplikacijski podatki (spremenljive dolžine)	

TCP: rušenje povezave

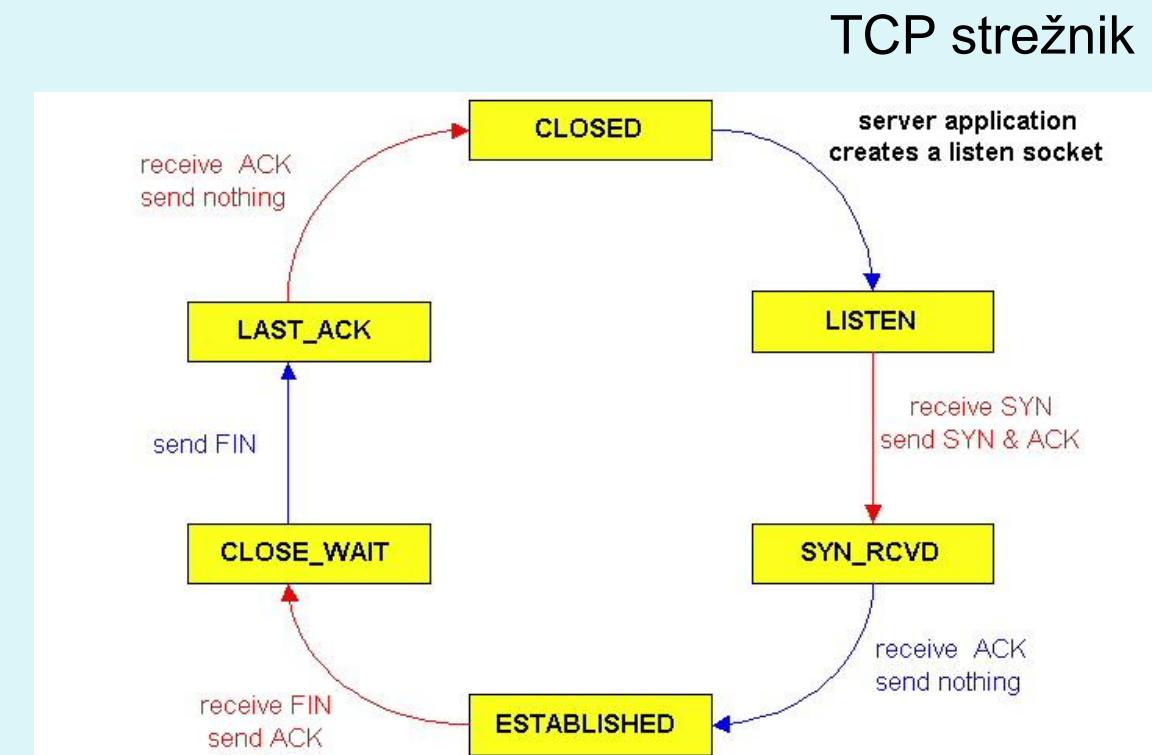
1. odjemalec pošlje segment TCP FIN strežniku
2. strežnik potrdi z ACK, zapre povezavo, pošlje FIN
3. odjemalec prejme strežnikov FIN, potrdi ga z ACK
 - počaka časovni interval, da po potrebi ponovno pošlje ACK, če se ta izgubi
4. strežnik sprejme ACK, končano



Življenjska cikla odjemalca in strežnika



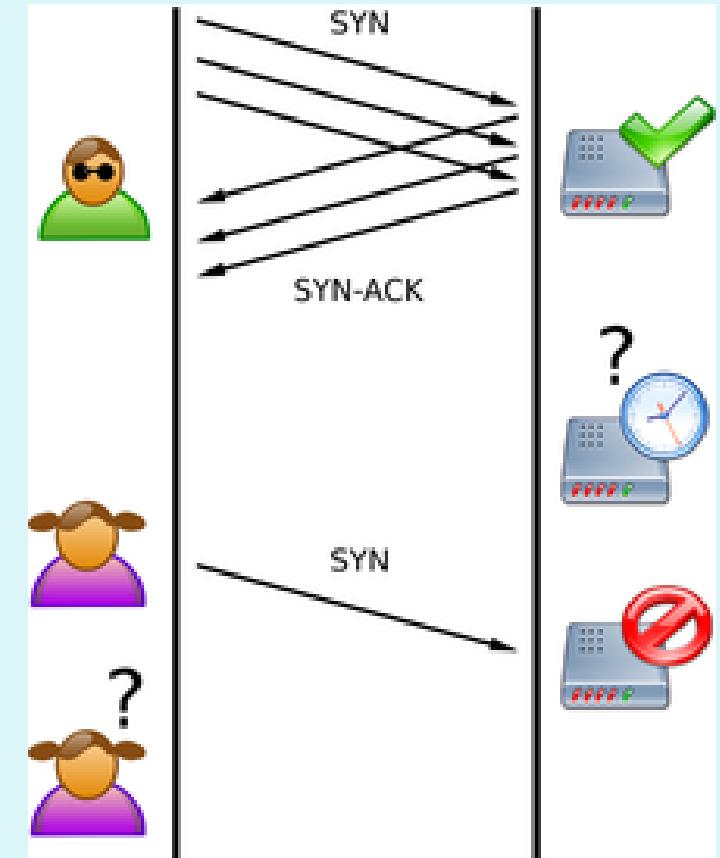
TCP odjemalec



TCP strežnik

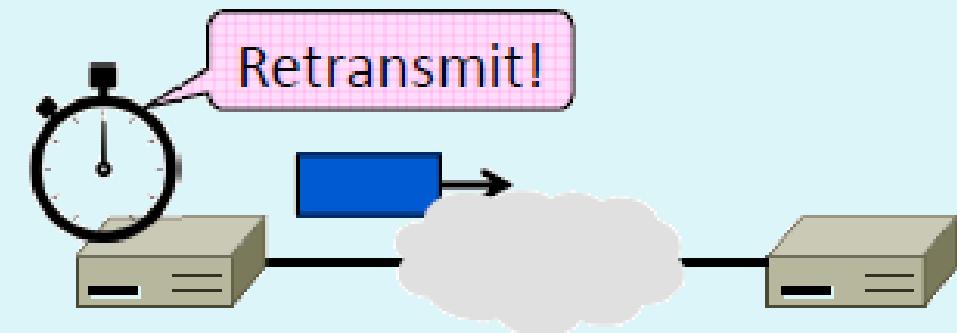
Varnost: napad SYN FLOOD

- 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 rokovana 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



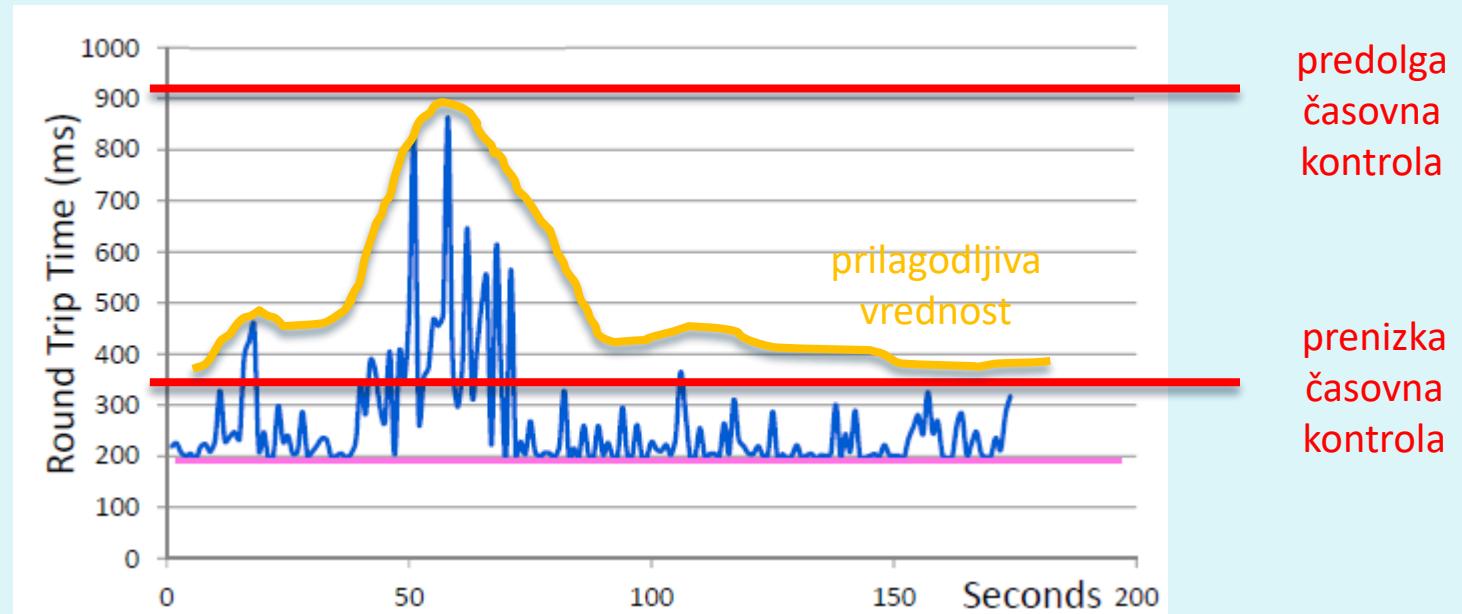
Nastavitev časovne kontrole

- **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

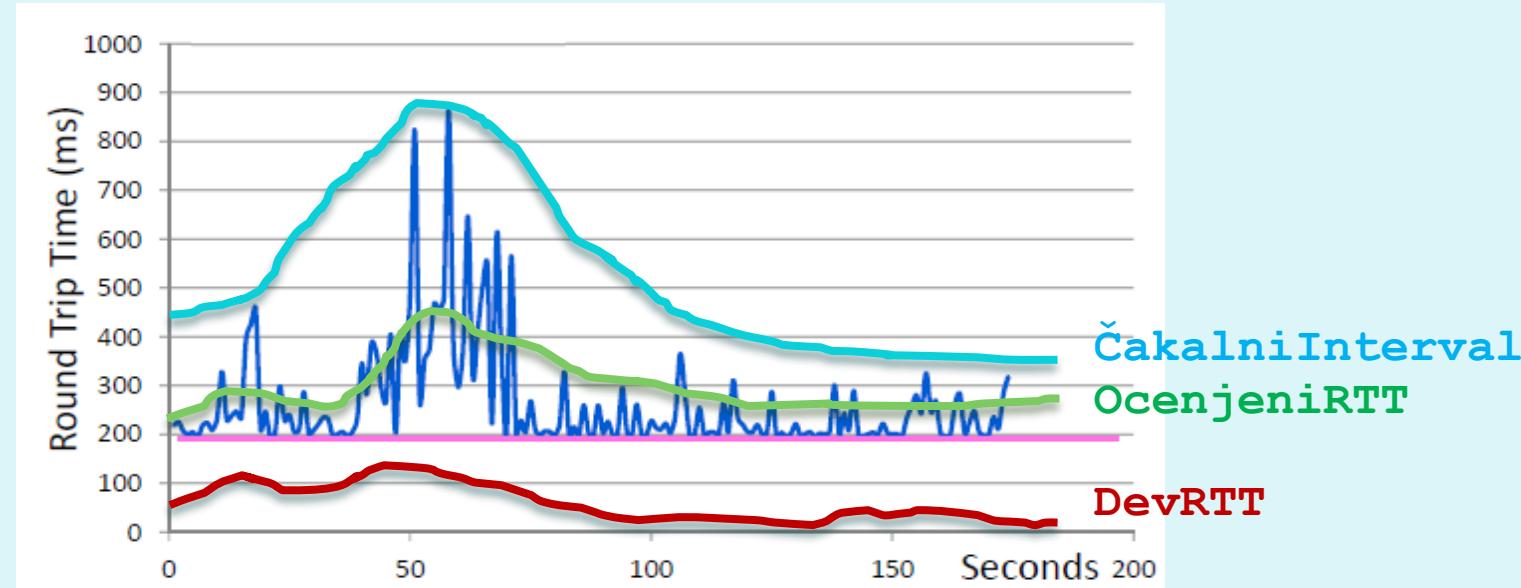


Primer ocenjevanja RTT

- 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
 $OcenjeniRTT[i] \leftarrow (1-\alpha) * OcenjeniRTT[i-1] + \alpha * IzmerjeniRTT[i]$
običajno uporabimo: $\alpha=0.125$
- izračunamo gibajočo deviacijo
 $DevRTT[i] = (1-\beta) * DevRTT[i-1] + \beta * |IzmerjeniRTT[i] - OcenjeniRTT[i]|$
običajno uporabimo $\beta=0.25$
- vrednost čakalnega intervala TCP nastavi kot OcenjeniRTT + "rezerva":
 $\checkmark \text{ČakalniInterval}[i] = OcenjeniRTT[i] + 4 * DevRTT[i]$

Način potrjevanja

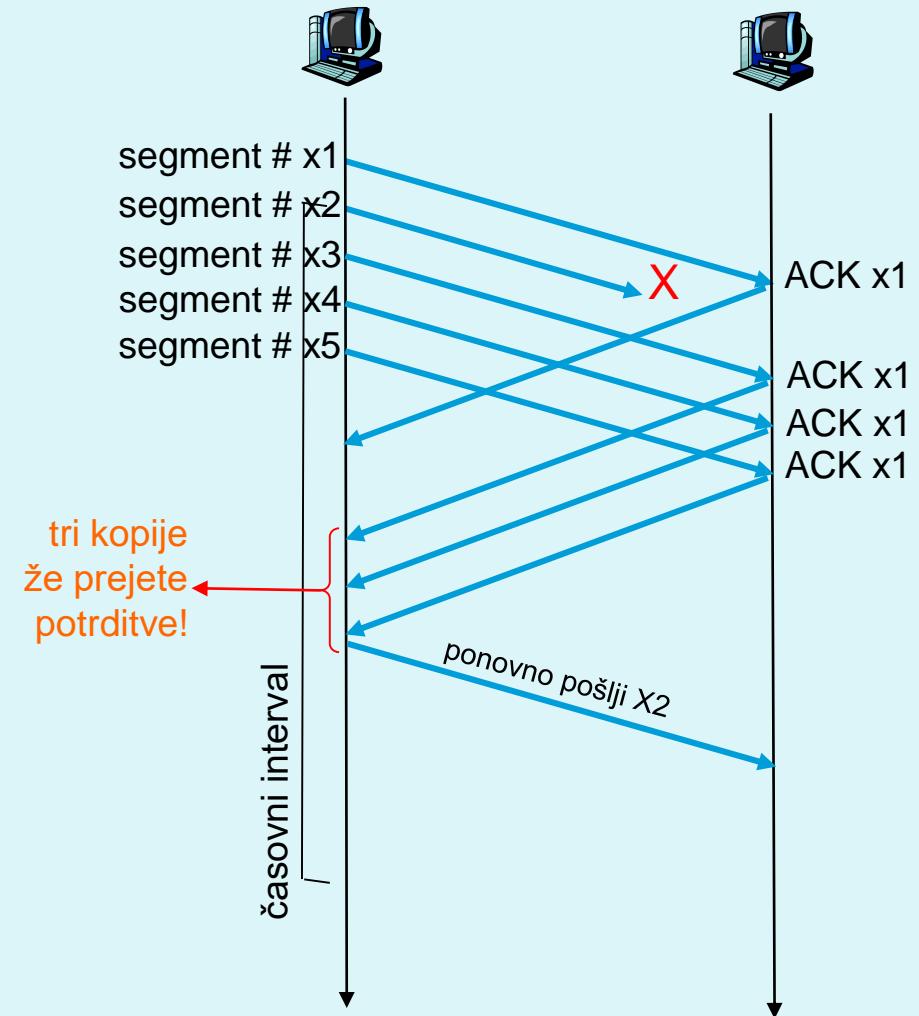
- katere vrste tekočega potrjevanja uporablja TCP?
 - ponavljanje N nepotrjenih (*go-back-N*)?
 - potrjevanje posameznih (*selective repeat*)?
- ODGOVOR: uporablja **kombinirano rešitev** obeh
 - podoben ponavljanju N nepotrjenih (štoparica za **najstarejši nepotrjeni segment**), vendar ob poteku časovne 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.	Tako 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.

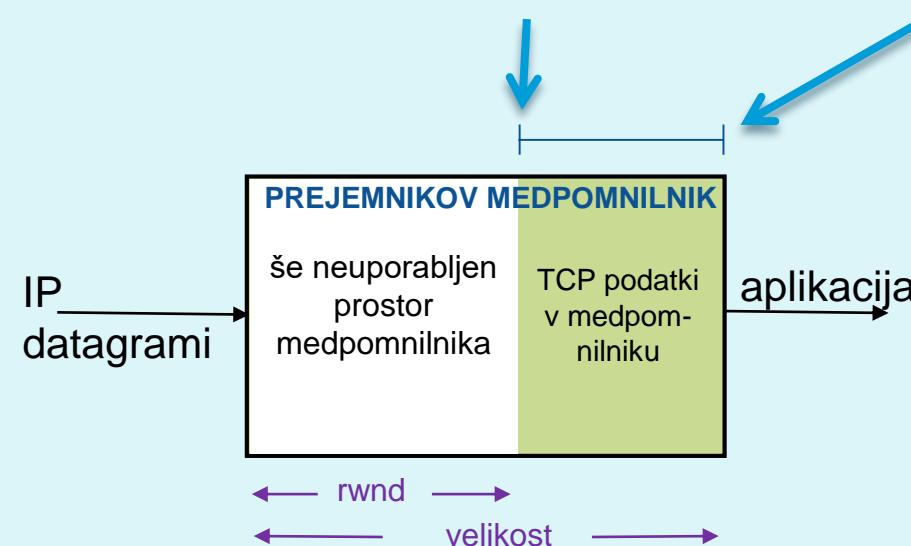
Hitro ponovno pošiljanje (*fast retransmit*)

- ponovno pošiljanje se običajno izvede **po preteku časovne kontrole**
- včasih je časovni interval predolг 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

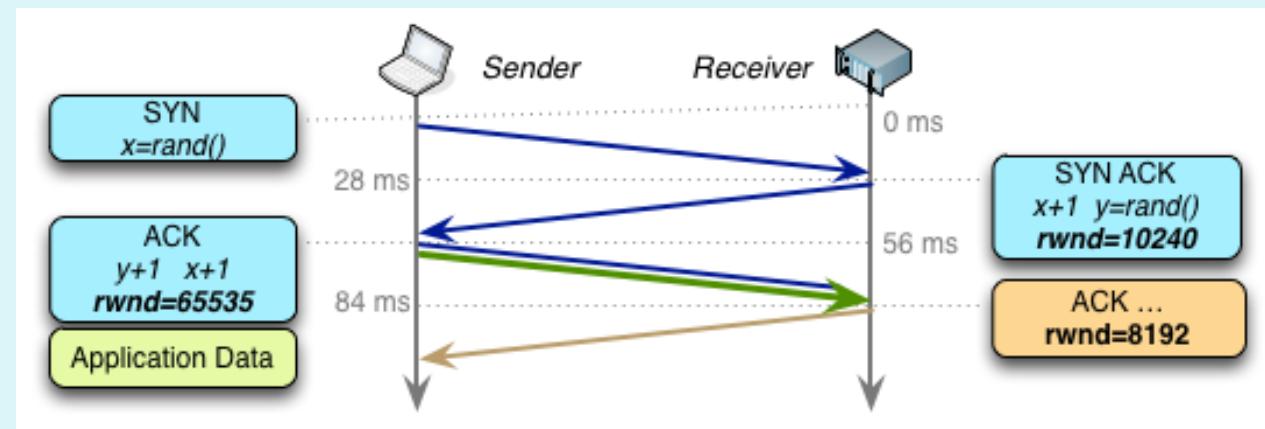
- uporablja se za usklajevanje hitrosti med pošiljateljem in prejemnikom: pošiljatelj ne sme pošiljati hitreje, kot lahko prejemnik bere, da ne povzroči prekoračitve medpomnilnika (prejemnikov prostor, kjer se začasno shranjujejo prejeti segmenti pred predajo aplikaciji)
- neuporabljen (razpoložljiv) prostor medpomnilnika:
$$\text{rwnd} = \text{velikost} - [\text{LastByteRcvd} - \text{LastByteRead}]$$



Kontrola pretoka TCP

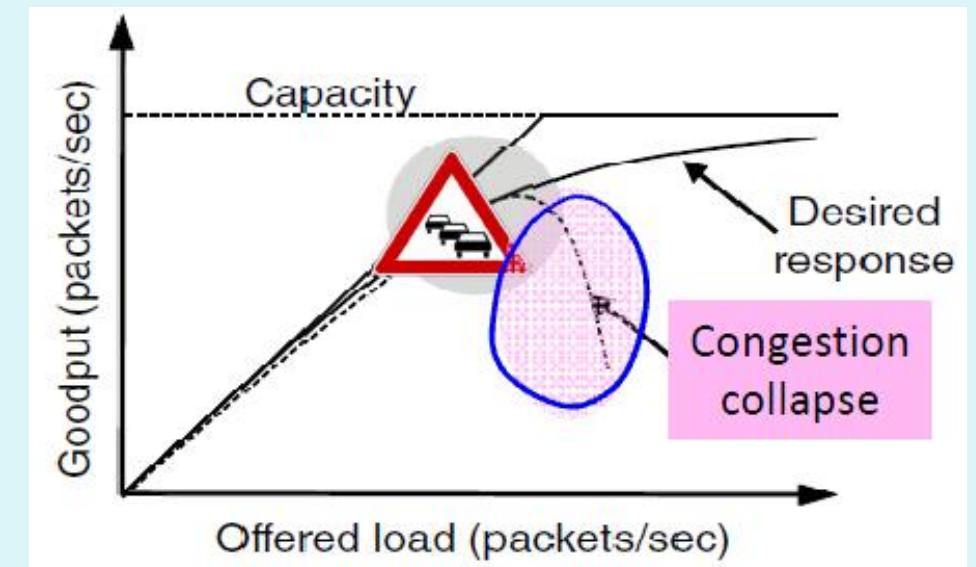
- prejemnik sporoča pošiljatelju velikost razpoložljivega prostora v glavi vsakega segmenta (rwnd)
- pošiljatelj ustrezeno omeji število paketov, za katere še ni prejel potrditve

št. izvornih vrat	št. ciljnih vrat
zaporedna številka	
številka potrditve	
dolžina glave	ni v uporabi
UA	P R S F
sprejemno okno	
kontrolna vsota	kazalec "urg data"
opcije (spremenljive dolžine)	
aplikacijski podatki (spremenljive dolžine)	



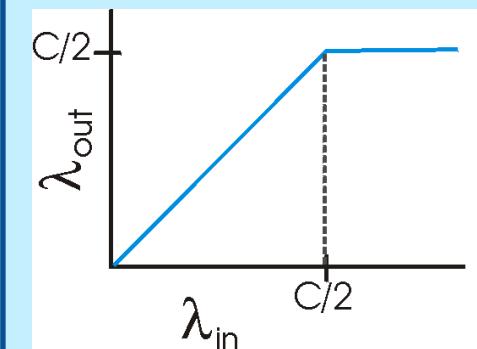
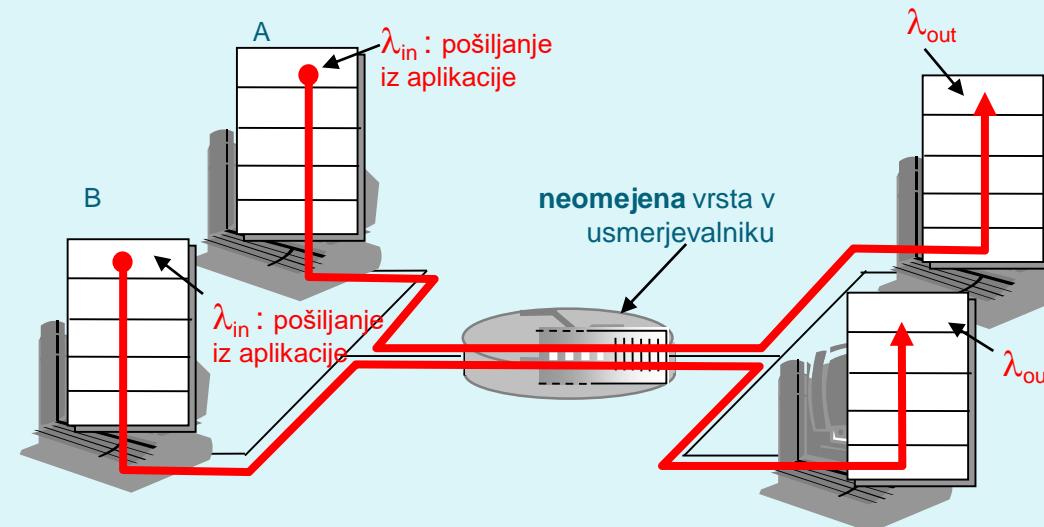
Nadzor zasičenja

- 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!



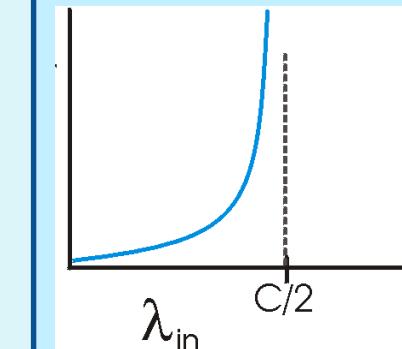
Zasičenje – primer 1

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



pretok:

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

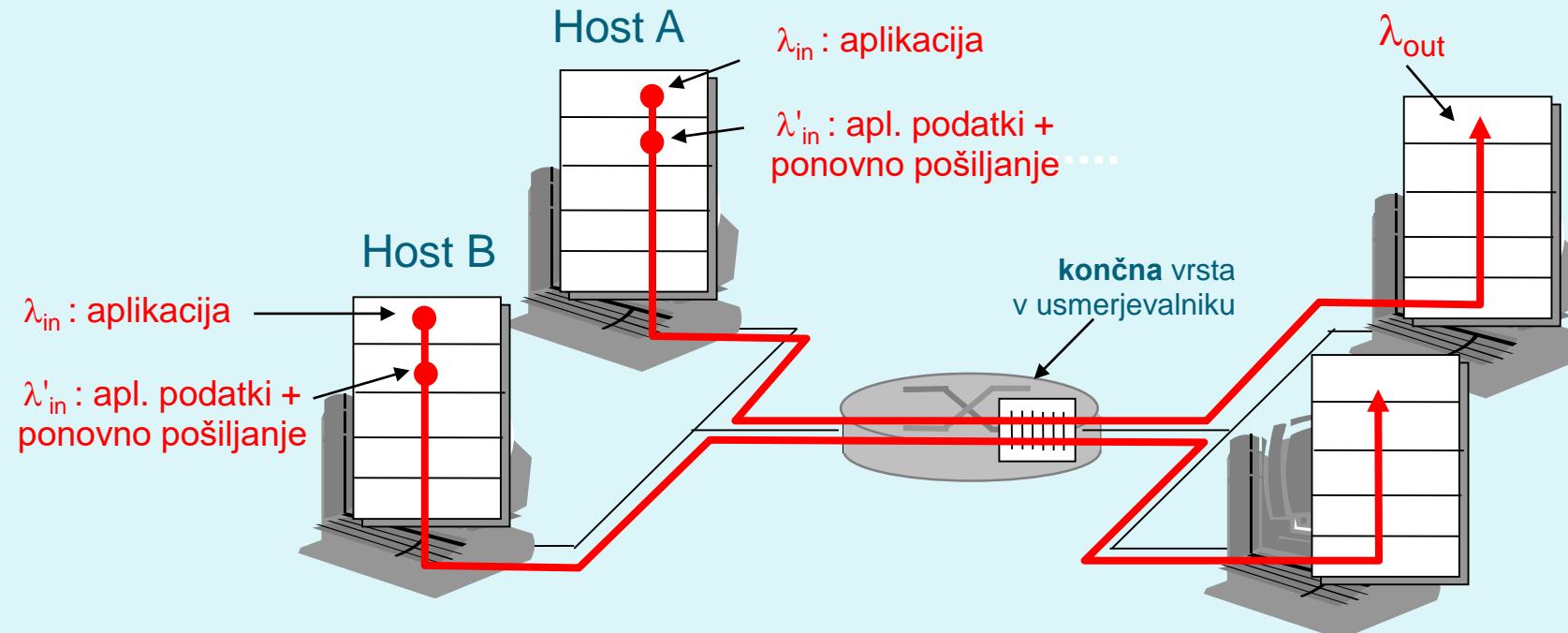


zakasnitev:

- 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

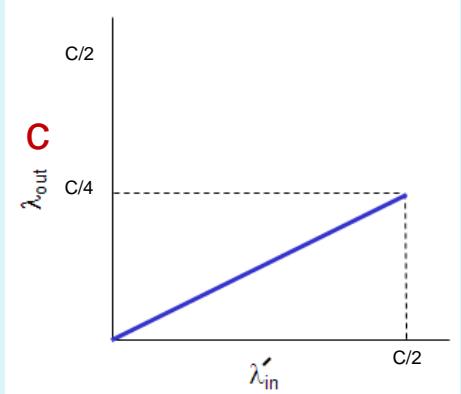
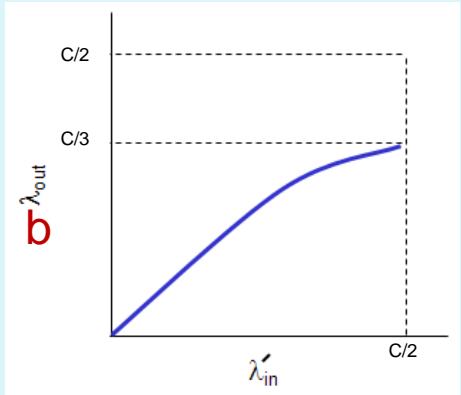
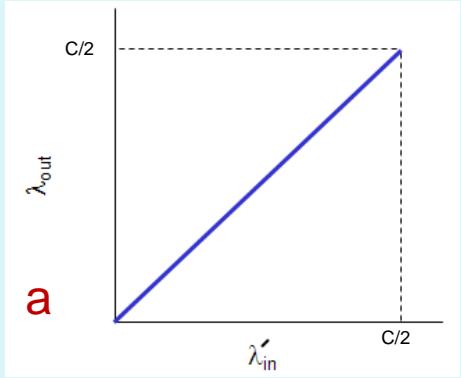


Zasičenje – primer 2

Preučimo tri scenarije:

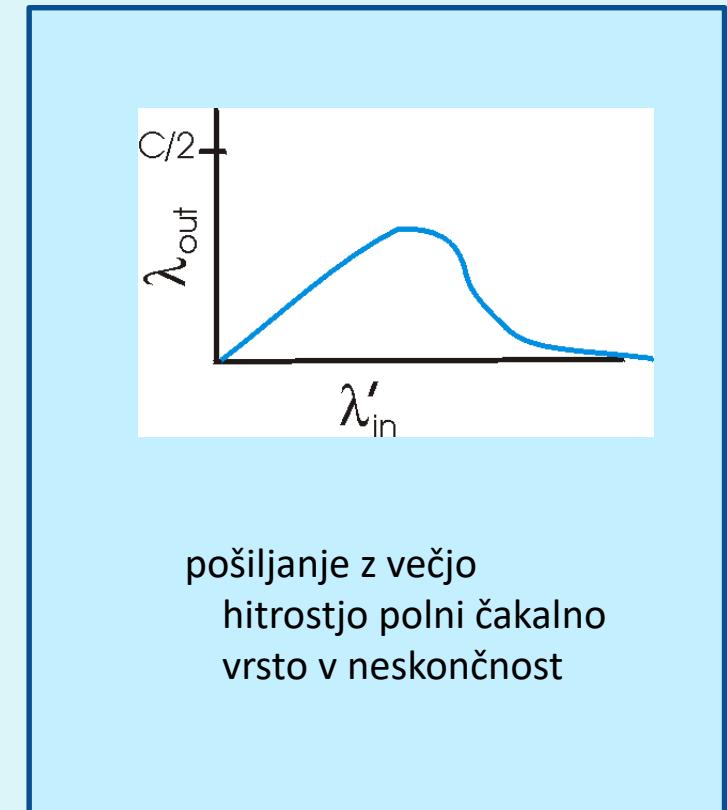
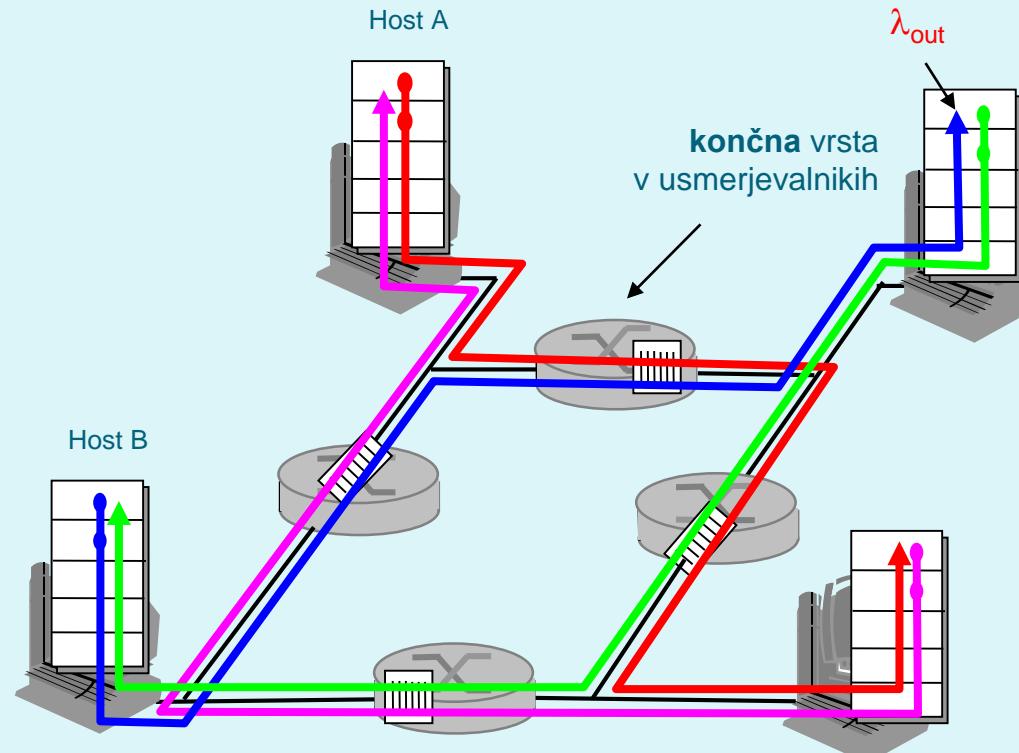
- a. segment oddamo le, ko je prostor v vrsti, tako da ni izgub
(v praksi to ni možno, ker tega ne vemo)
- b. dogajajo se izgube paketov in ponovna pošiljanja
- c. ponovna pošiljanja tudi zaradi velikih zakasnitev

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



Zasičenje – primer 3

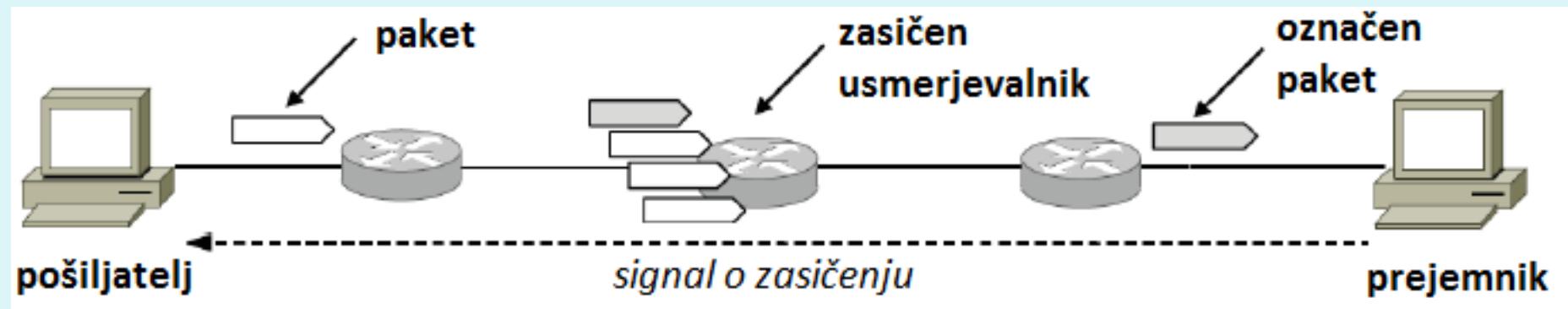
- daljše poti: če se paket izgubi na n -tem skoku, so bili zaman vsi dotedanji prenosi!



Kako nadzorovati zasičenja?

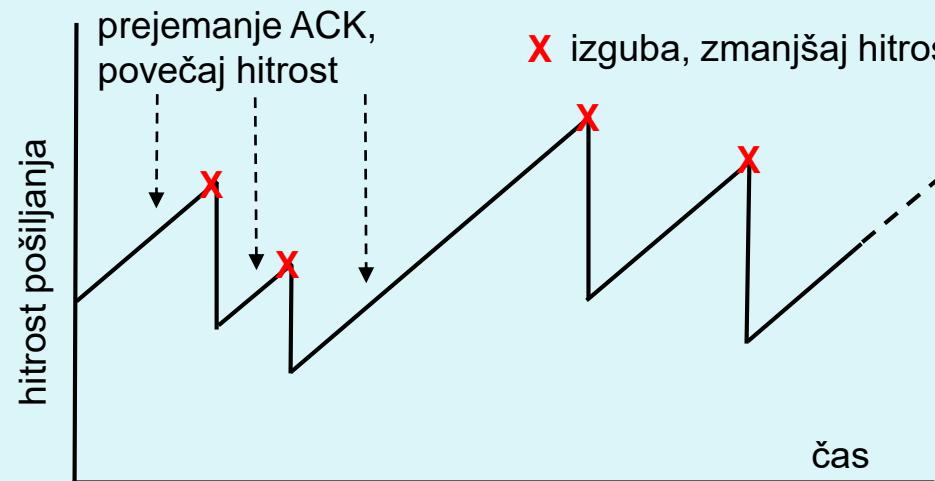
dva pristopa:

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

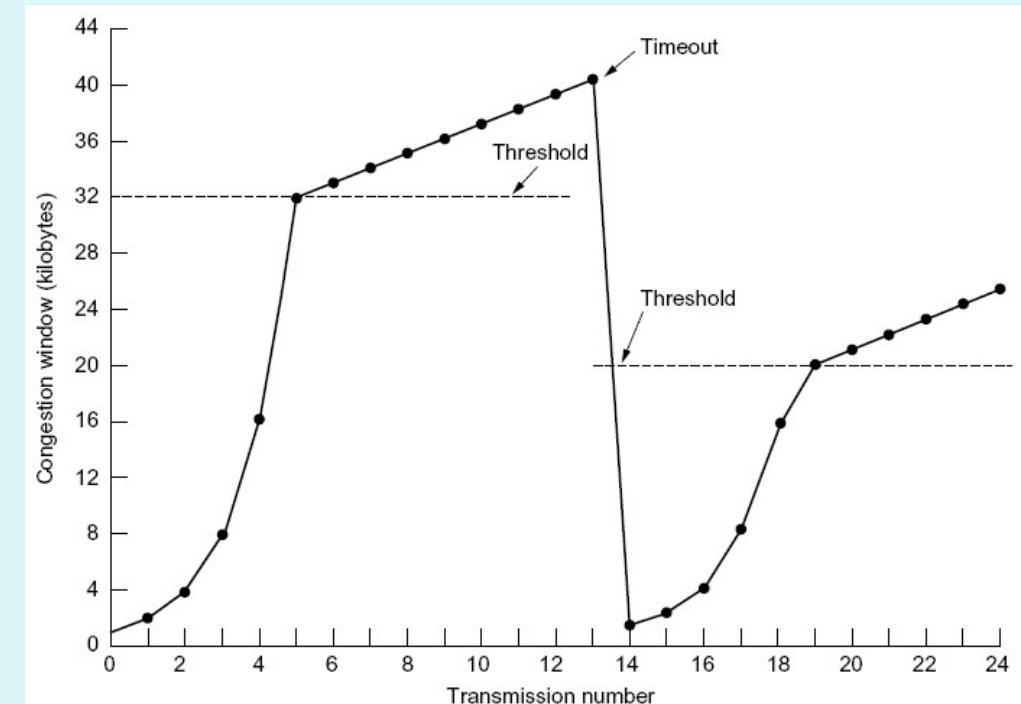
- **ideja:** pošiljatelj želi pošiljati ČIM HITREJE, vendar še POD MEJO zasičenja omrežja.
Kako najti pravo hitrost?
- **rešitev:** vsak pošiljatelj si sproti nastavlja hitrost na podlagi opazovanja reakcij v omrežju na pošiljanje:
 - če prejme potrditev (ACK), ni zasičenja, poveča hitrost
 - če se segment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



TCP ima
"žagasto
obliko" hitrosti
pošiljanja

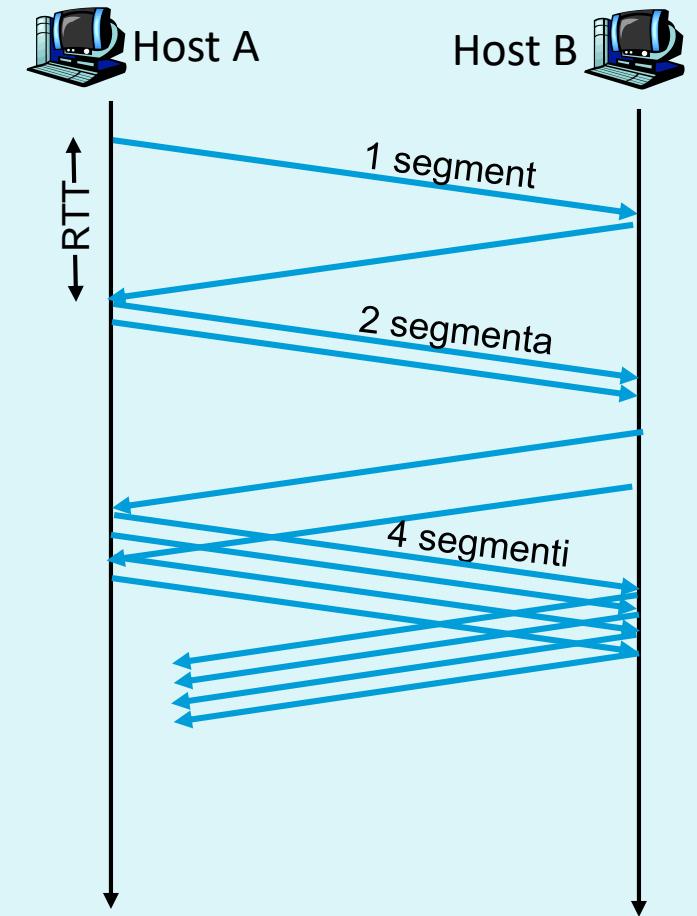
TCP nadzor zasičenja

- 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 ($\times 2$, **počasni začetek**, *slow start*) ali
 - linearno (+1, **izogibanje zasičenju**, *congestion avoidance*)
 - NEGATIVEN:**
potek časovnega intervala (segment se izgubi):
zmanjšaj cwnd na 1



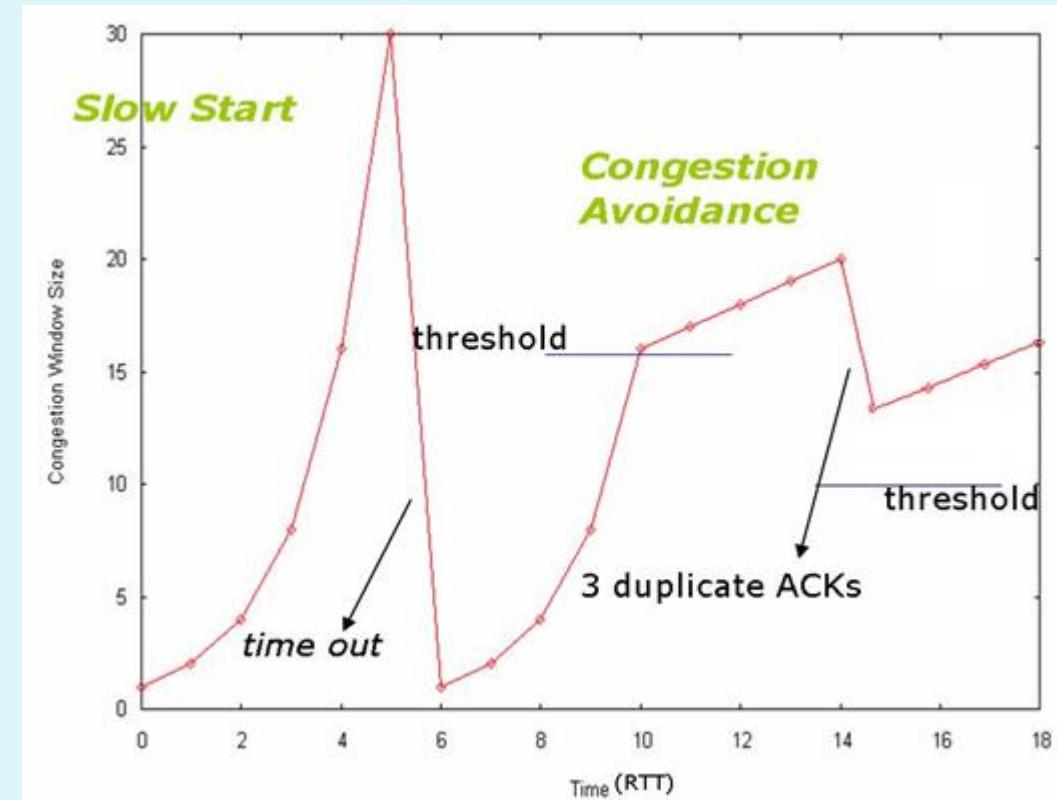
Počasen začetek (*Slow Start*)

1. ob vzpostavitvi povezave:
velja **cwnd = 1** segment
2. hitrost povečuj eksponentno:
 - za vsak prejeti ACK: $cwnd \leftarrow cwnd + 1$
 - (ozioroma po vseh prejetih ACK izgleda kot eksponentna rast
 $cwnd \leftarrow cwnd * 2$)
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**.

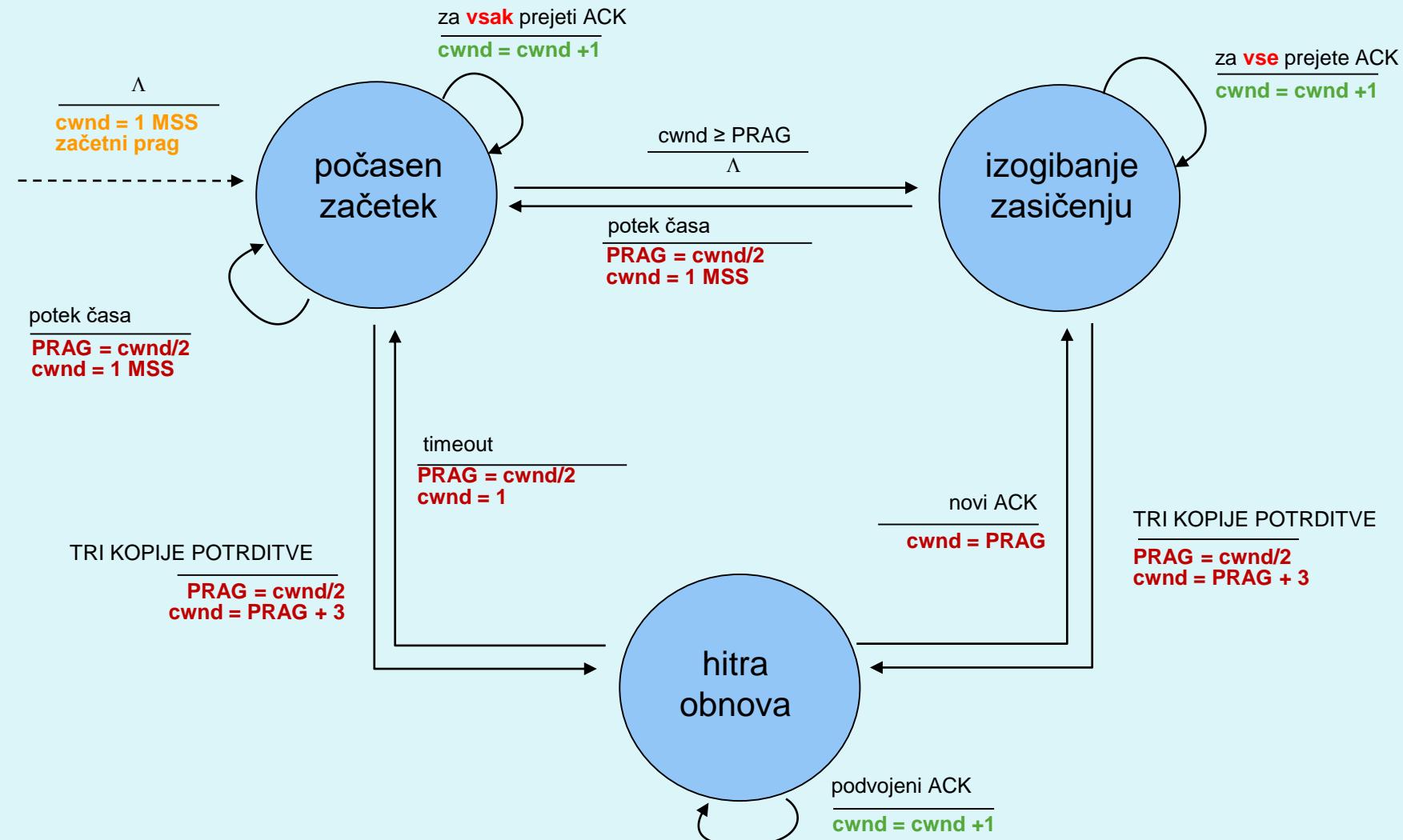


Izogibanje zasičenju (Congestion Avoidance)

- kadar **cwnd \geq PRAG**, povečuj cwnd linearno za 1 MSS
- na ta način se bolj počasi približaj pragu zasičenja
- negativni dogodki:
 - **potek časovne kontrole**: cwnd $\leftarrow 1$, 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 \leftarrow cwnd/2
 - cwnd \leftarrow cwnd/2 + 3

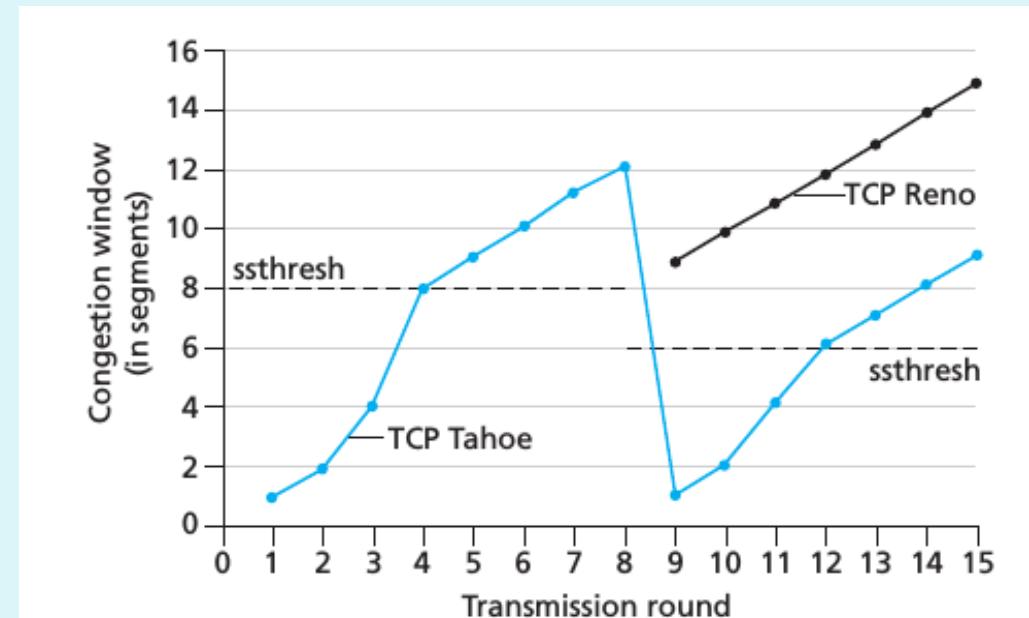


Končni avtomat za TCP nadzor zasičenja

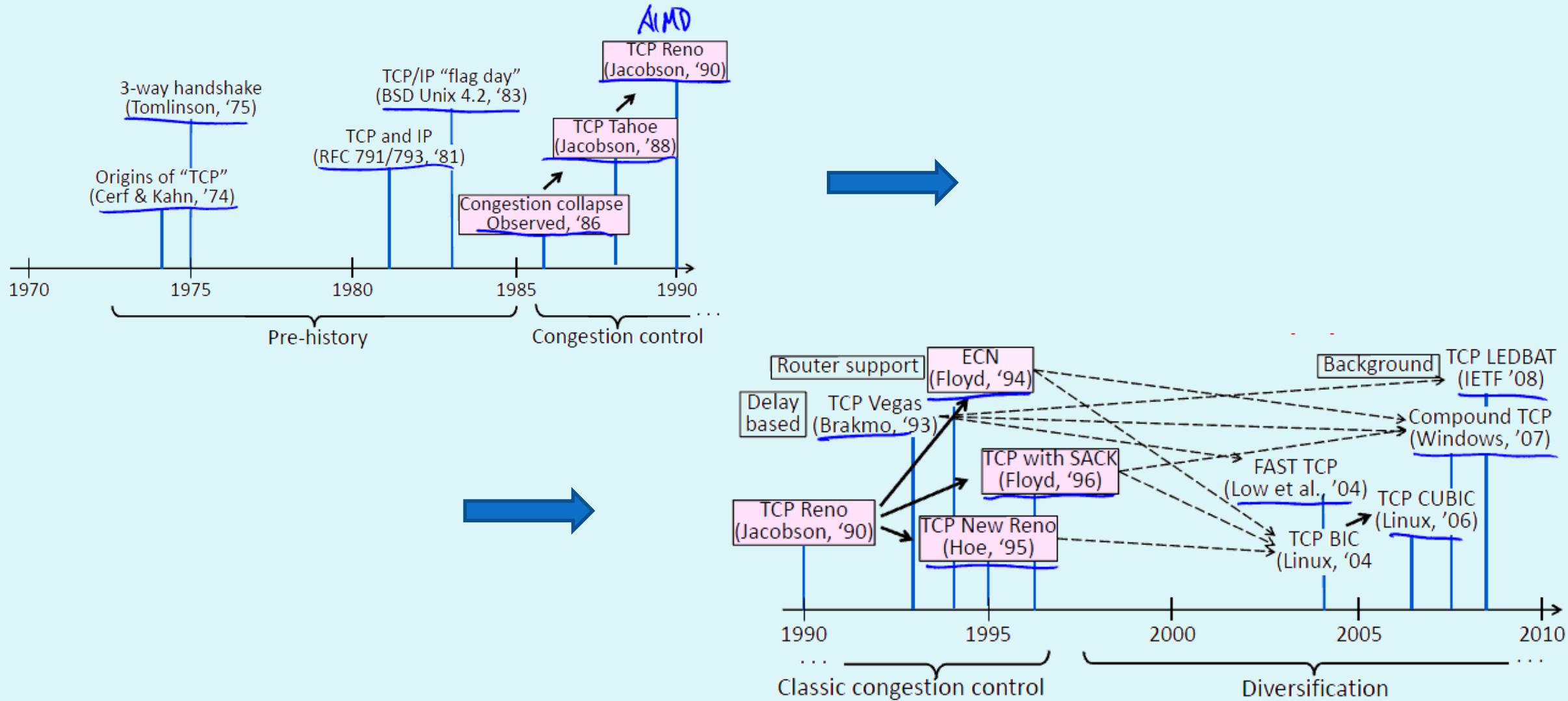


Razvoj nadzora zasičenja skozi različice TCP

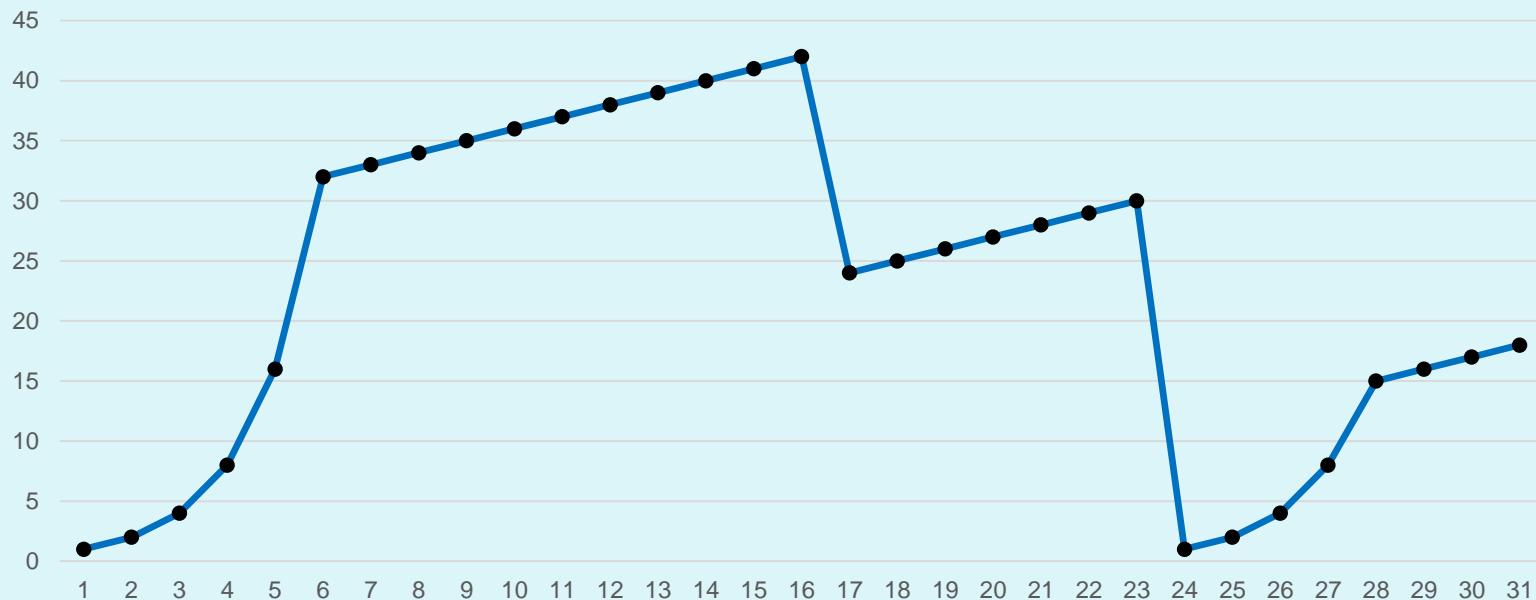
1. **TCP Tahoe**: osnovna verzija (uporablja samo *počasen začetek* in *izogibanje zasičenja*), po izgubi paketa vedno nastavi cwnd=1
2. **TCP Reno**: dodana faza *hitre obnove* - po prejemu treh kopij iste potrditve, preskoči počasen začetek in nastavi $cwnd \leftarrow cwnd/2 + 3$
3. **TCP Vegas**: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



Zgodovina razvoja TCP



Primer: analiza delovanja TCP Reno

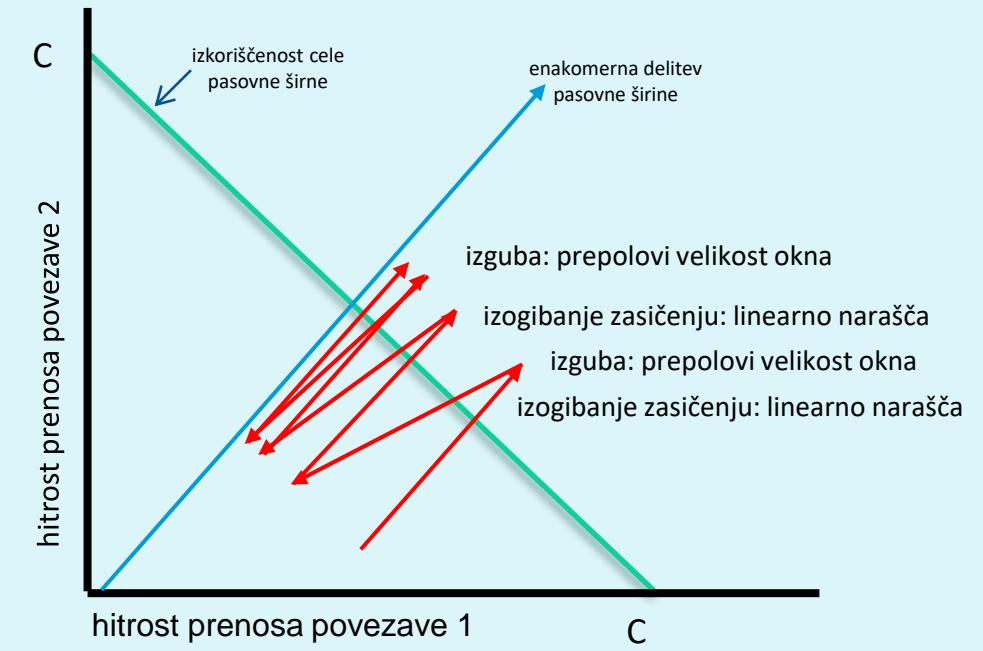
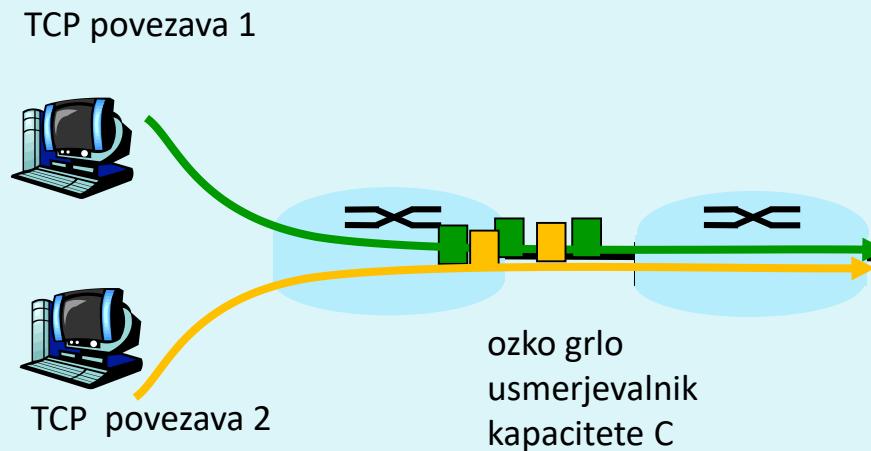


Primeri vprašanj za analizo delovanja protokola:

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?

- 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

Naslednje poglavje?

- aplikacijska plast!

