

# Računalniške komunikacije

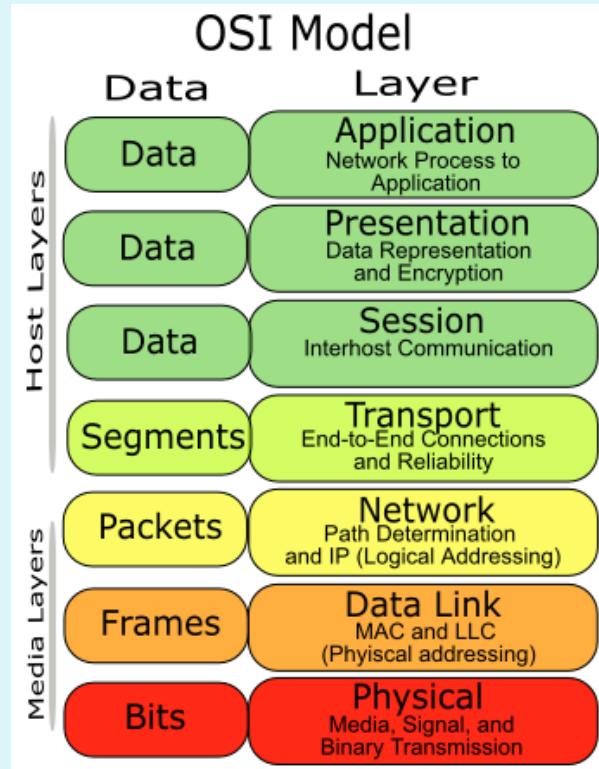
2020/21

transportna plast  
storitve, multipleksiranje,  
protokol UDP, konstrukcija TCP, potrjevanje

# Pridobljeno znanje s prejšnjih predavanj

- **protokol IPv6**
  - **motivacija za uvedbo**
    - večji naslovni prostor (128-bitni naslovi)
    - hitrejše usmerjanje (fiksna glava paketa, ukinitev opcij fragmentacije in kontrolne vsote)
    - zagotavljanje kakovosti storitev (oznaka za vrsto toka)
  - **prehodni mehanizmi**
    - dvojni sklad (uporaba odgovorov DNS strežnikov za identifikacijo IPv6, izguba oznake toka pri konverziji IPv6->IPv4)
    - tuneliranje (enkapsulacija IPv6 v IPv4)
- **usmerjanje**
  - pojmi: cena povezave, najcenejša pot
  - **centralizirani algoritmi** (gradijo drevo najkrajših poti)
  - **decentralizirani (podazdeljeni) algoritmi** (usmerjanje z vektorji razdalj)
    - posredovalne tabele se računajo iterativno z minimizacijo vsote cene do sosedja in ocenjene cene sosedove razdalje do cilja
    - principa: good news travel fast, bad news travel slow

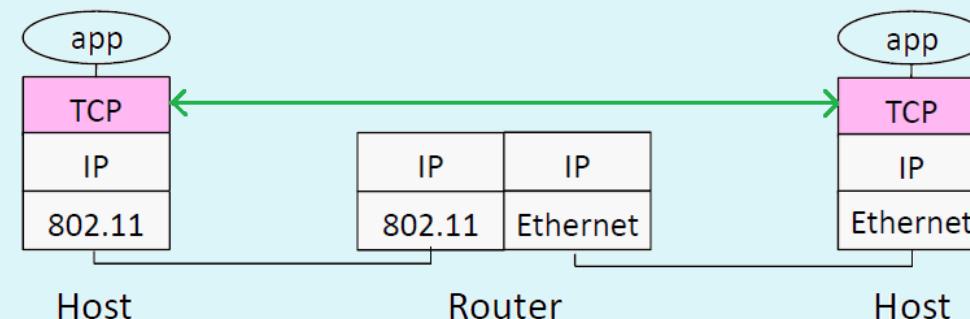
# Transportna plast



## Naloge transportne plasti:

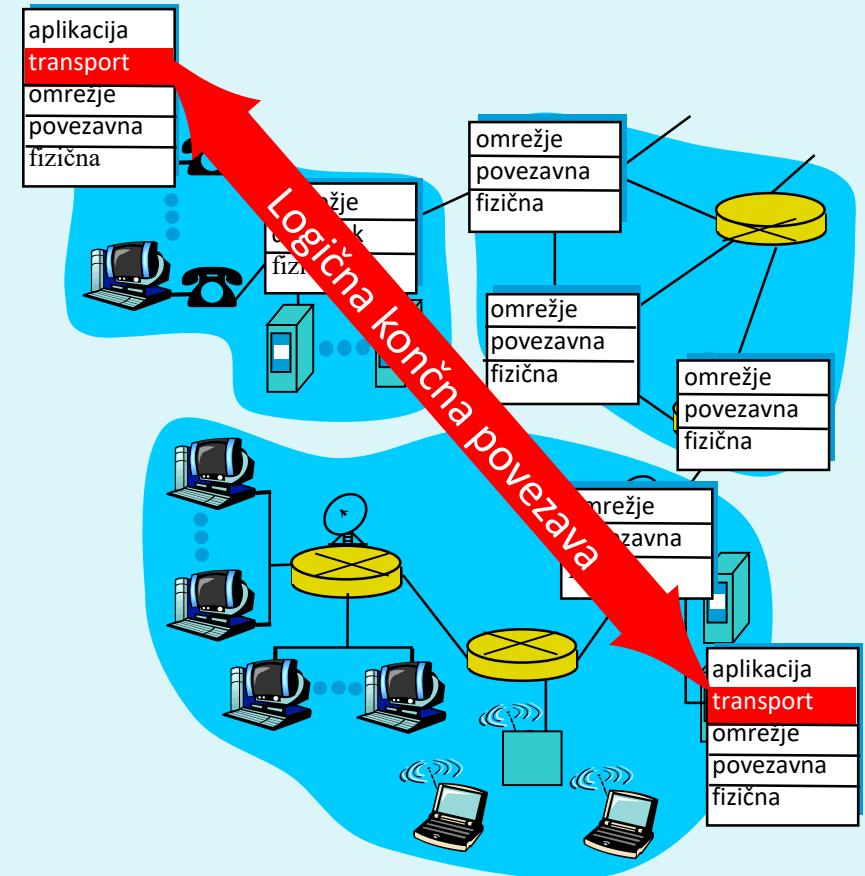
- povezovanje dveh oddaljenih **procesov**
- multipleksiranje/demultipleksiranje komunikacije med procesi
- zanesljiv prenos podatkov
- kontrola pretoka in zasičenja

TCP: segment  
UDP: datagram



# Storitve transportne plasti

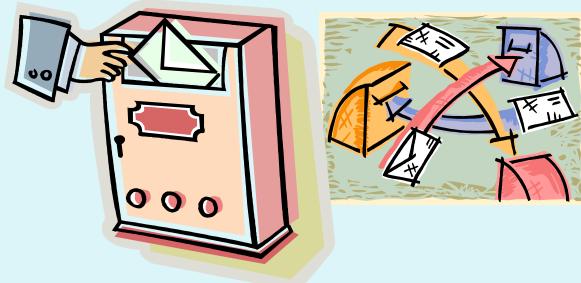
- Logična komunikacija med aplikacijskimi procesi
  - **pošiljatelj**: sporočilo razbije v SEGMENTE in jih posreduje v enkapsulacijo omrežni plasti
  - **prejemnik**: dekapsulira segmente iz paketov, sestavljene segmente združi v sporočila in jih posreduje aplikacijski plasti
- Protokola TCP in UDP



# Analogija



Ana



Eva

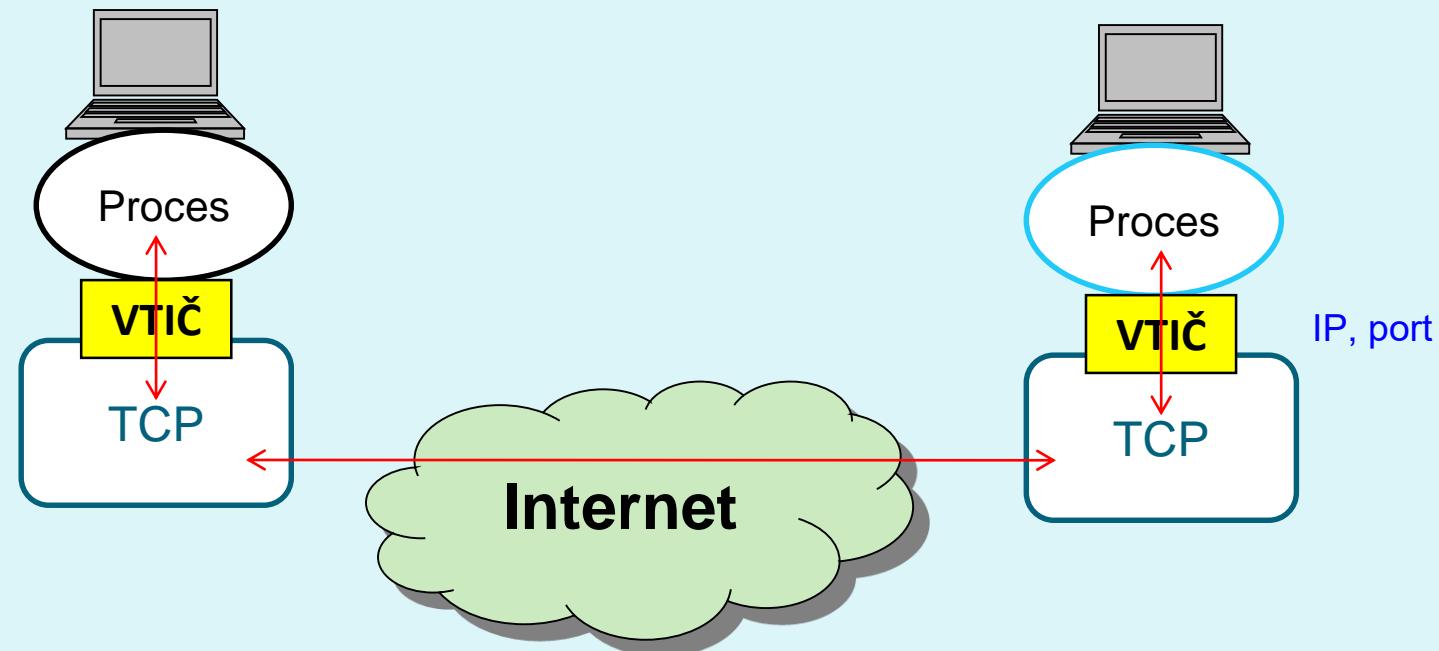
končni sistem = hiša  
proces = otrok  
aplikacijsko sporočilo = pismo  
transportni protokol = Ana in Eva  
omrežna plast = storitev pošte

# 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

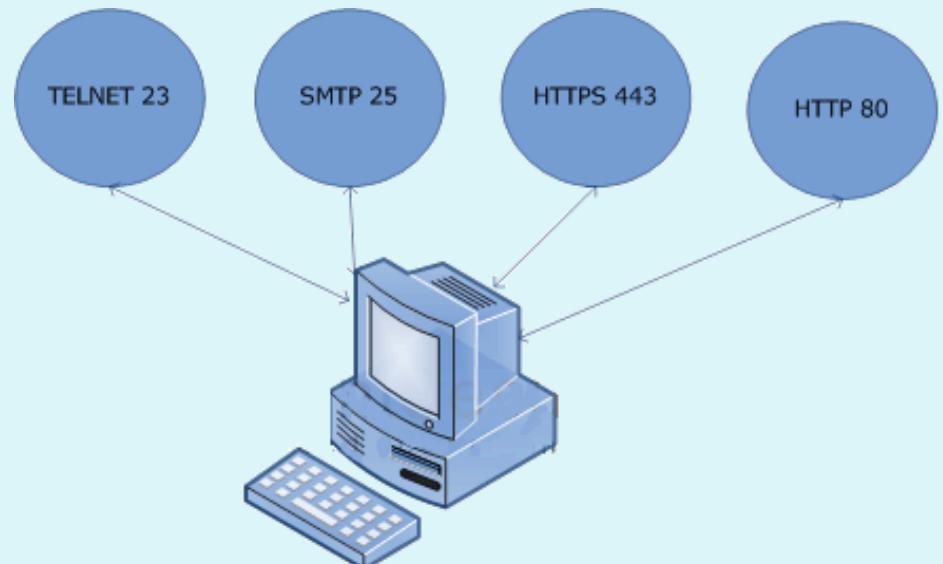
# Kako komunicirati s procesom (aplikacijo)?

- 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 nasloviti proces na drugi strani?

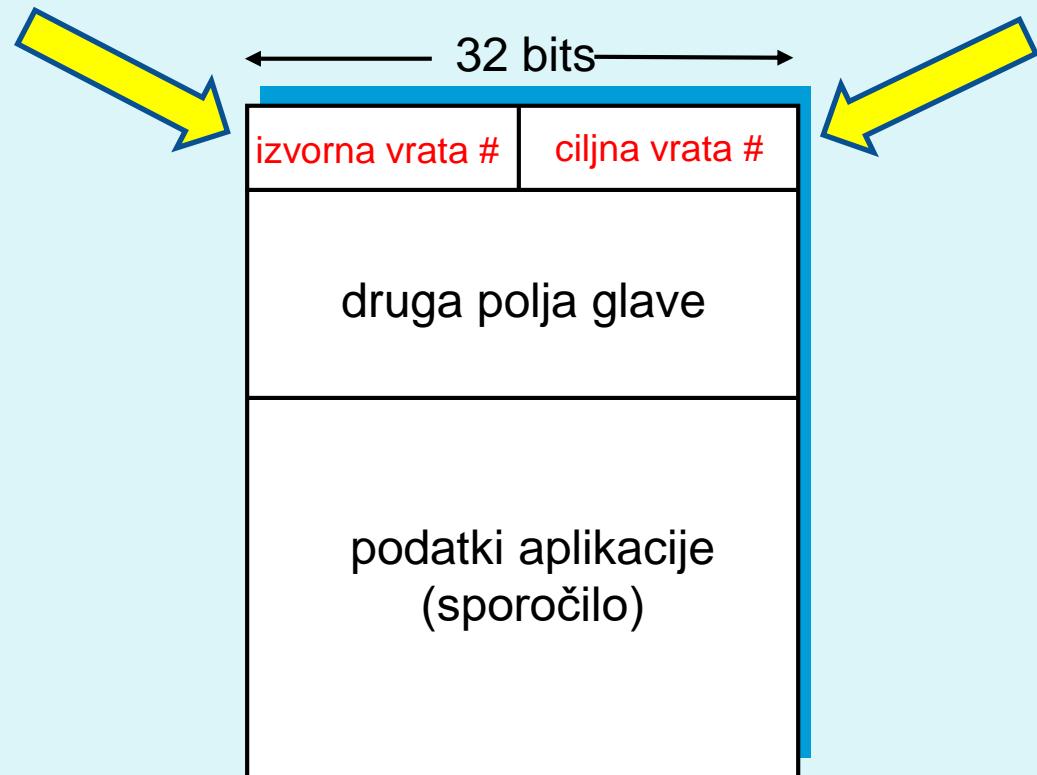
- za naslovitev vtiča potrebujemo:
  - naslov vmesnika naprave (host address): IP številka
  - naslov procesa (znotraj naprave): številka vrat ([port](#))
- znane aplikacije uporabljajo znane številke vrat 0-1023 (t.i. well-known ports), npr.
  - spletni strežnik (HTTP): 80
  - poštni strežnik (SMTP): 25
  - imenski strežnik (DNS): 53
  - oddaljen dostop (telnet): 23
  - pogovorni strežnik (IRC): 194
  - več: [www.iana.org](http://www.iana.org)



# Kako poteka demultiplexiranje?

Port je 16-bitna številka (0 - 65535)

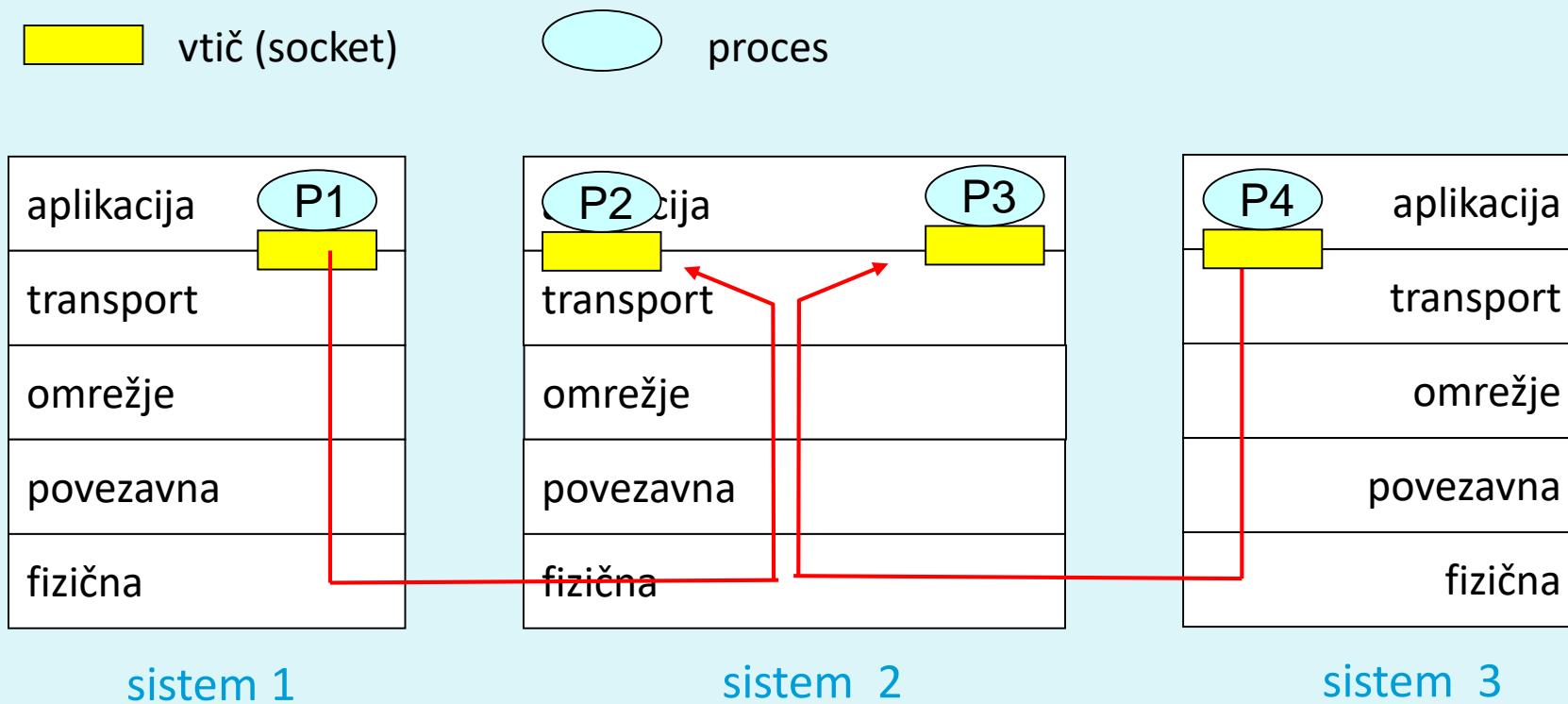
- vsak transportni segment potuje znotraj svojega paketa IP
- **transportni segment ima 16-bitna podatka:** številka vrat *izvora* (oznaka procesa, ki pošilja) in *ponora* (oznaka procesa na ciljni strani)



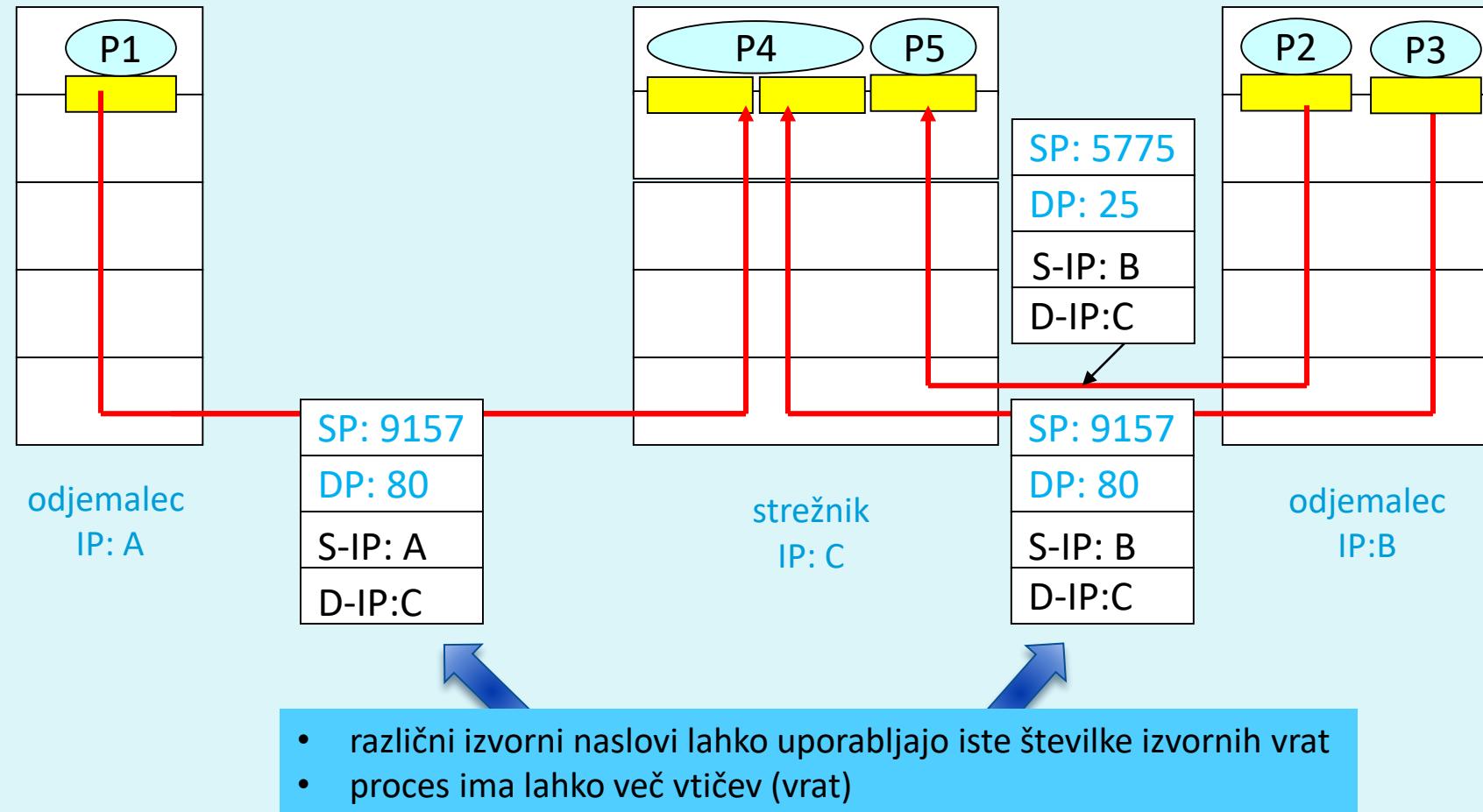
oblika segmenta TCP/  
datagrama UDP

# Multipleksiranje in demultipleksiranje

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



# Povezavno demultiplexiranje (TCP)

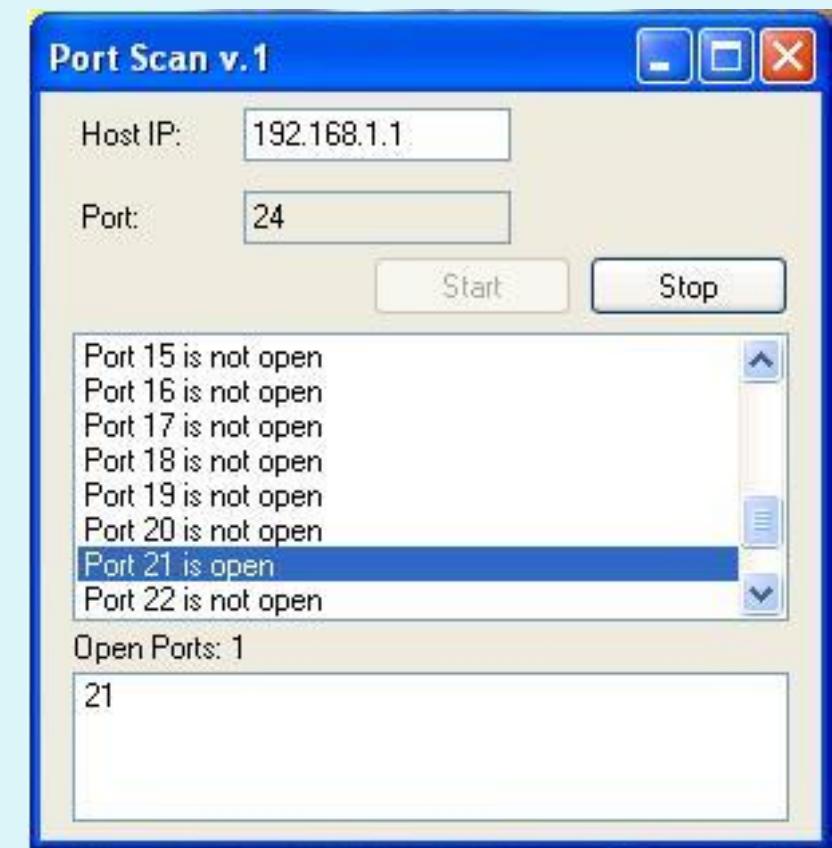


SP = source port, izvorna vrata (predstavlja naslov za odgovor)

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.

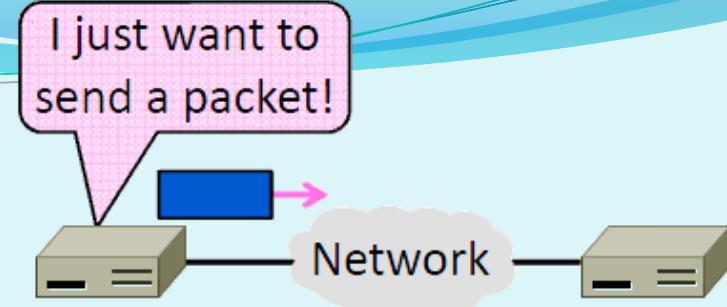


# Nepovezavni transport: UDP

*(User Datagram Protocol)*



# Storitve protokola UDP



## LASTNOSTI

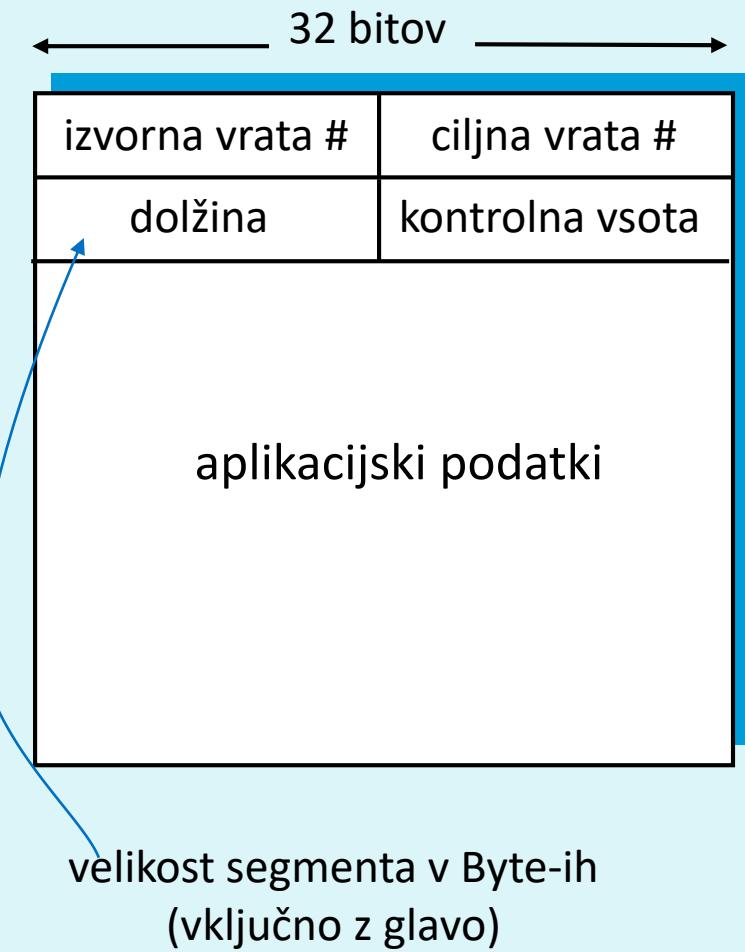
- nudi le "best-effort" storitev:
  - izgubljeni datagrami ("segmenti")
  - ne zagotavlja vrstnega reda
- nepovezaven (nima rokovanja)
- nima nadzora zamašitev

## 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

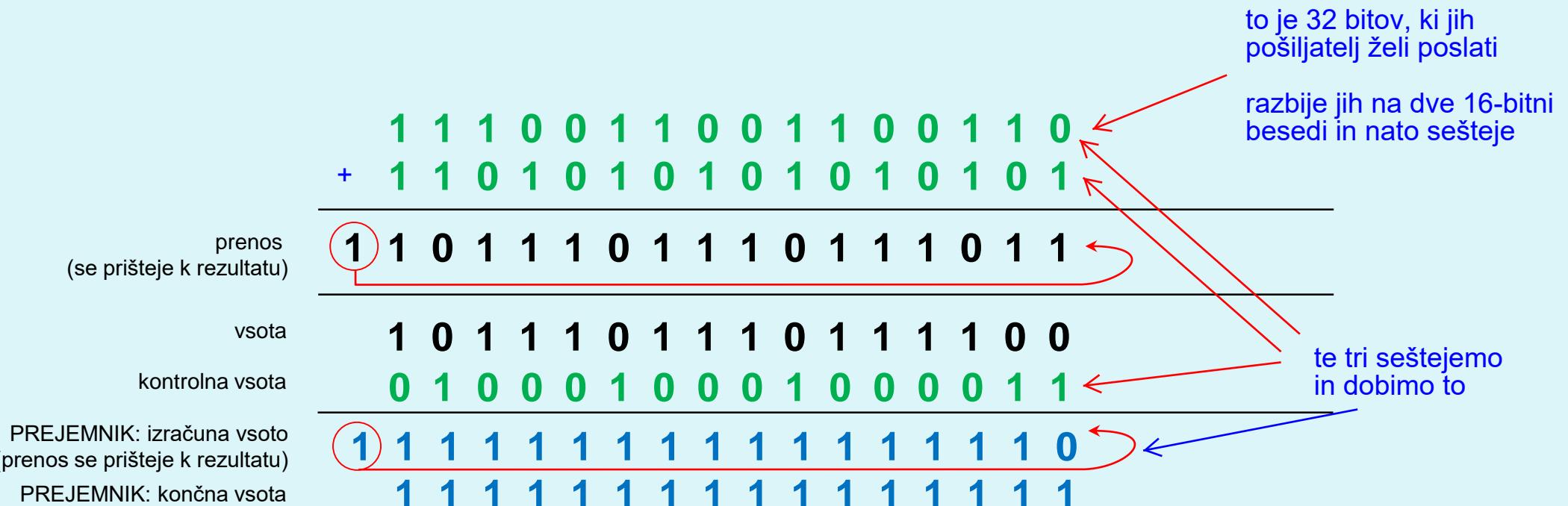
# UDP datagram

- 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

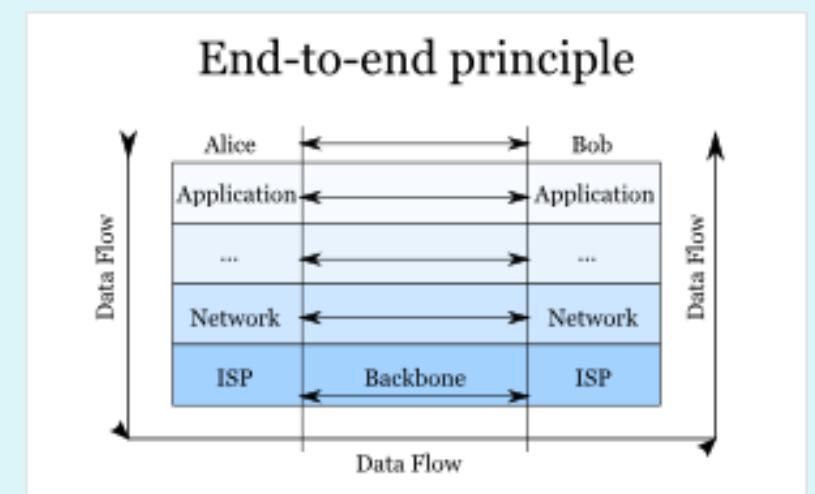
- algoritem za izračun imenujemo internetna kontrolna vsota (*Internet Checksum*):
  - pošiljatelj** sešteje 16 bitne besede in shrani eniški komplement = kontrolna vsota
  - prejemnik** sešteje 16 bitne besede skupaj s kontrolno vsoto -> dobiti mora same enice



# UDP: internetna kontrolna vsota

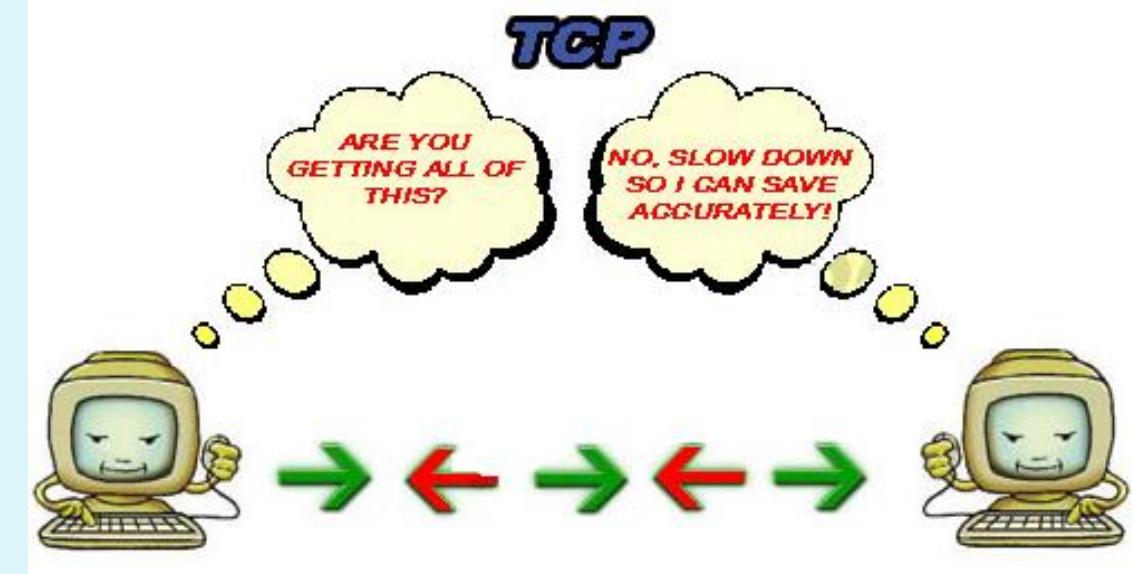
- zakaj datagram vsebuje kontrolno vsoto?

- 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*?



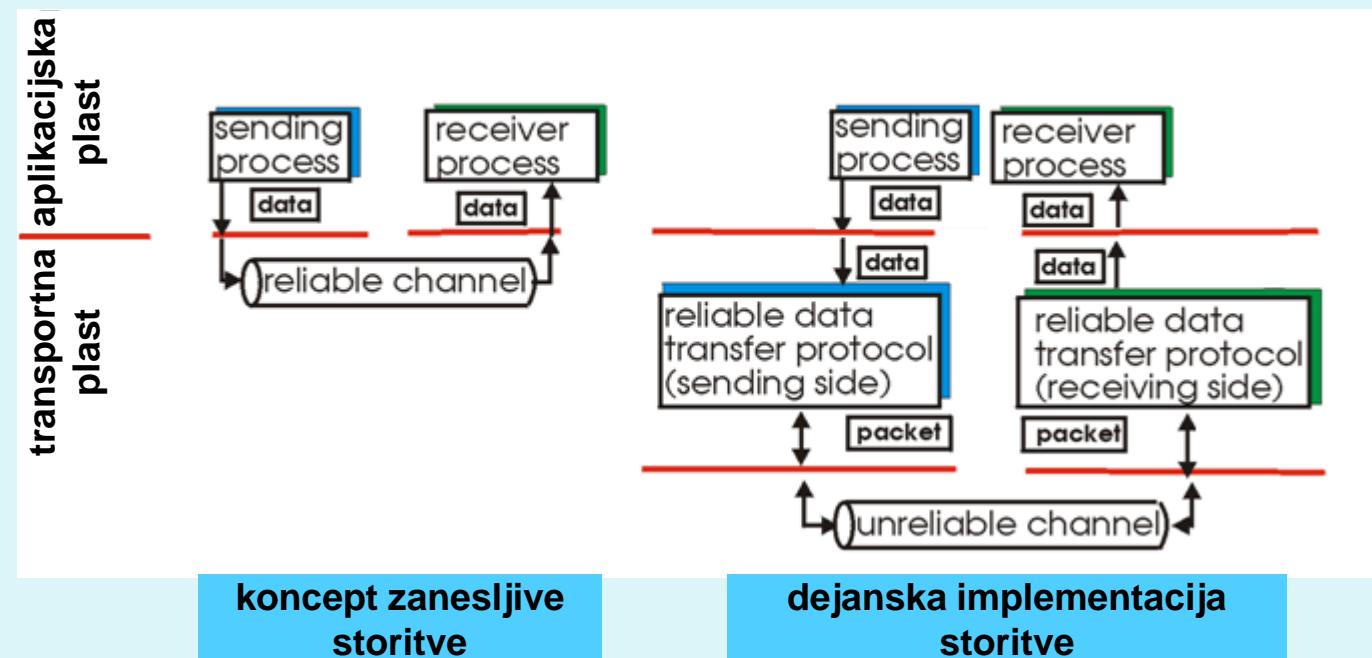
# Povezavni transport: TCP

*(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

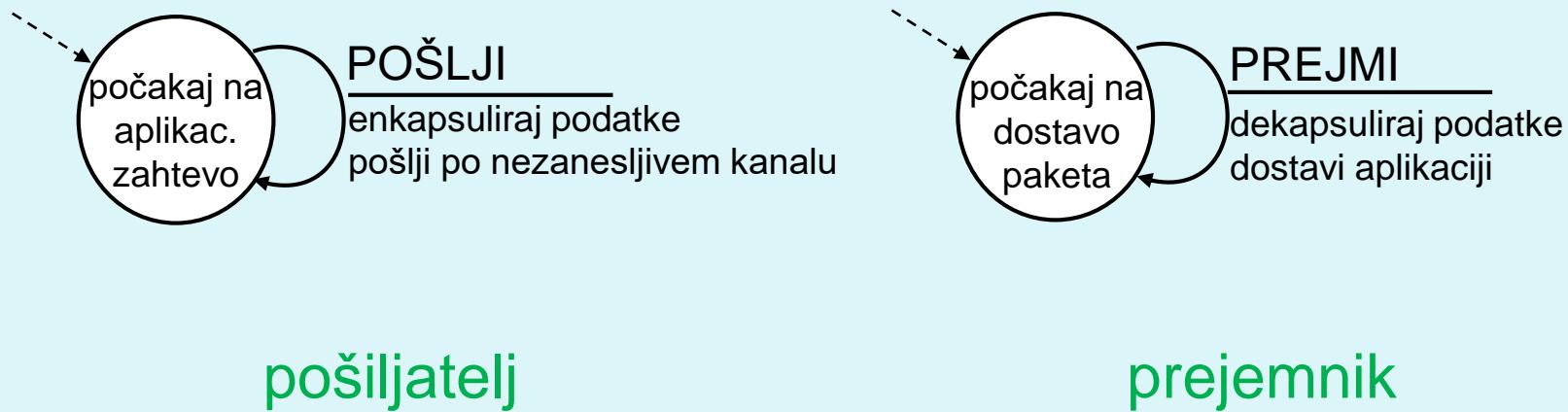


# Protokol TCP

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

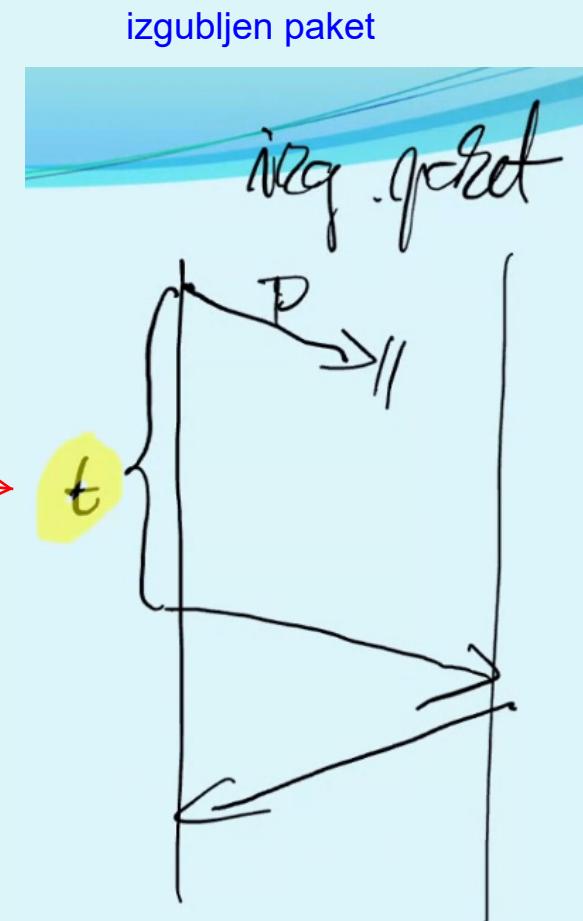
# 1. Enostavni protokol

- 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)

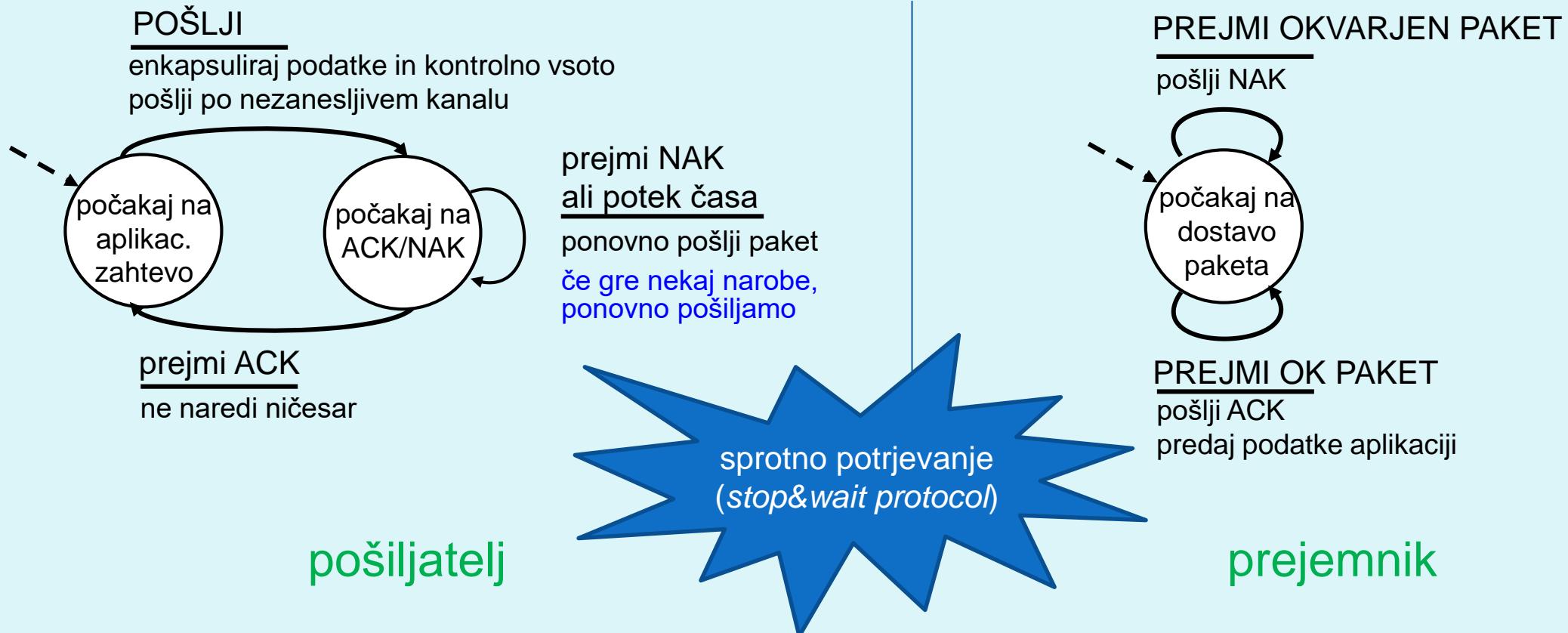


## 2. Reševanje iz napak pri prenosu

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

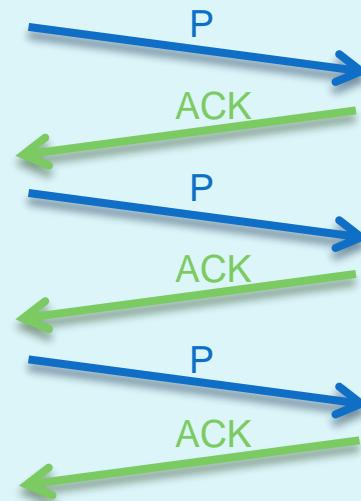


## 2. Reševanje iz napak pri prenosu

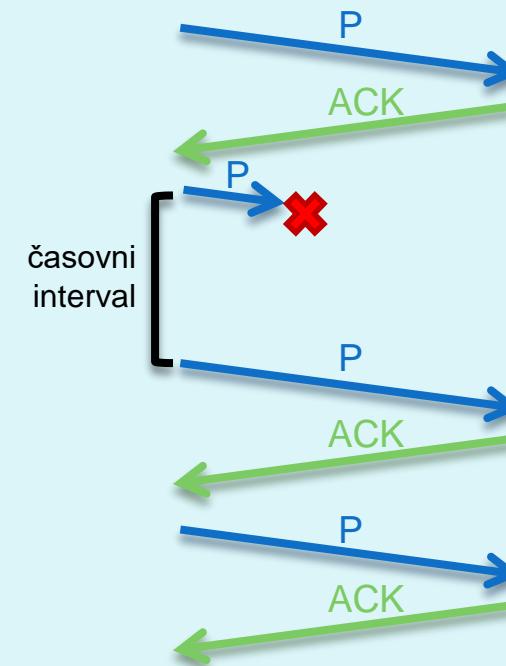


### 3. Izguba paketov ali potrditev

- 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



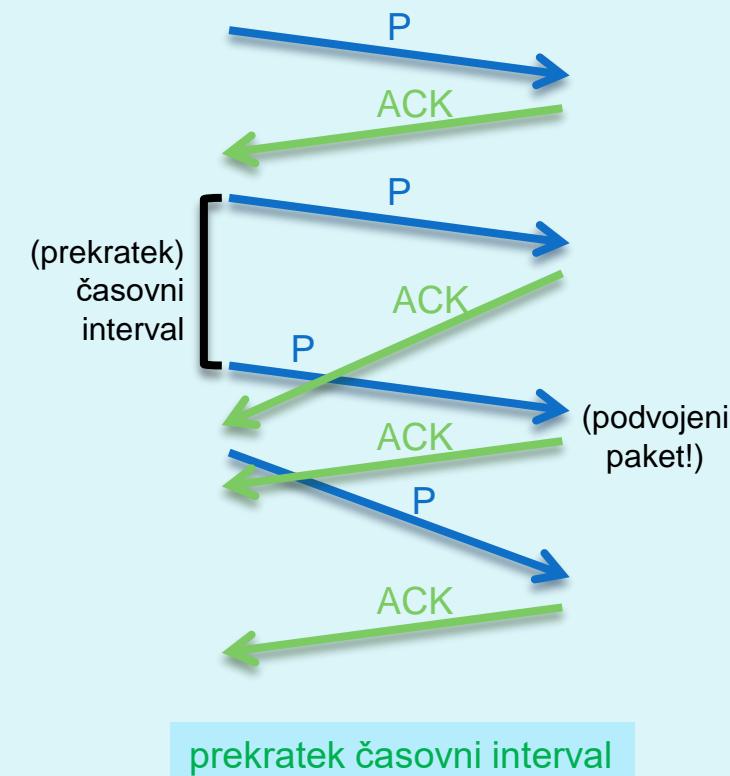
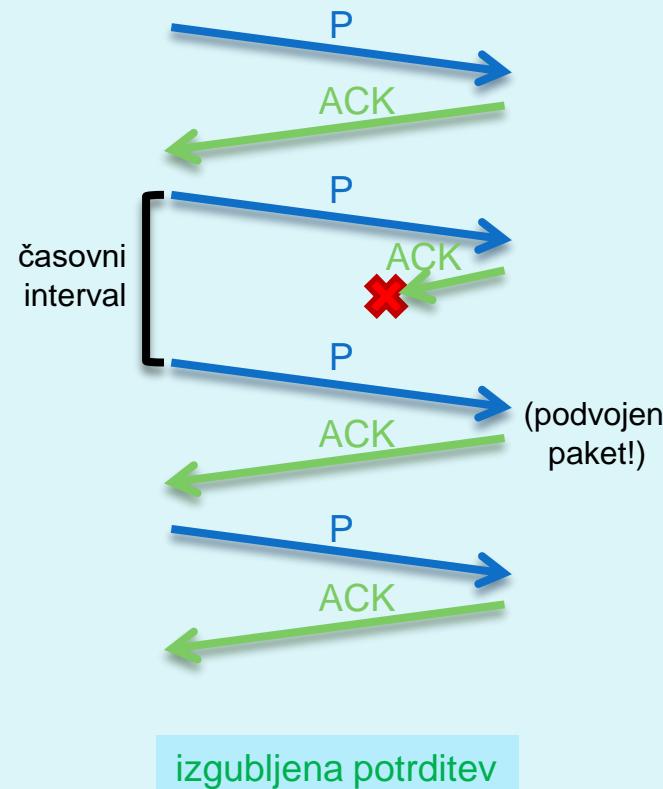
brez izgube paketa



z izgubo paketa

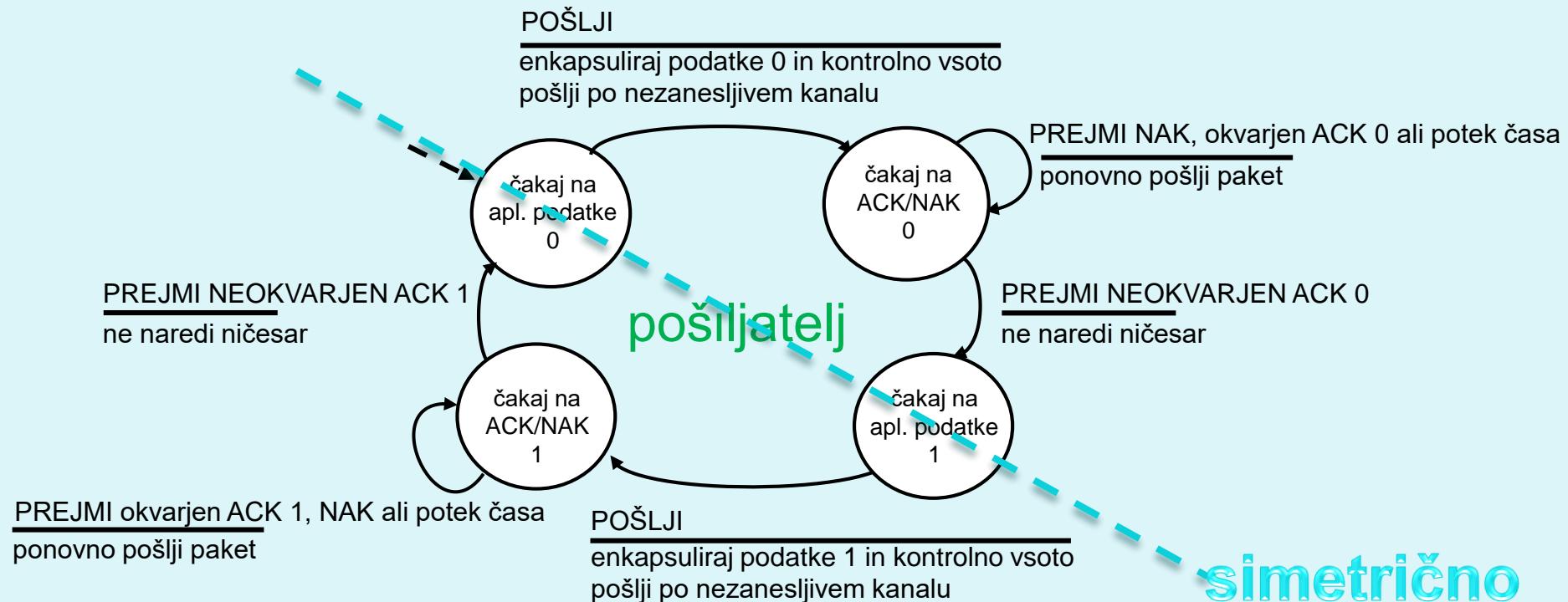
### 3. Izguba paketov ali potrditev

- 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

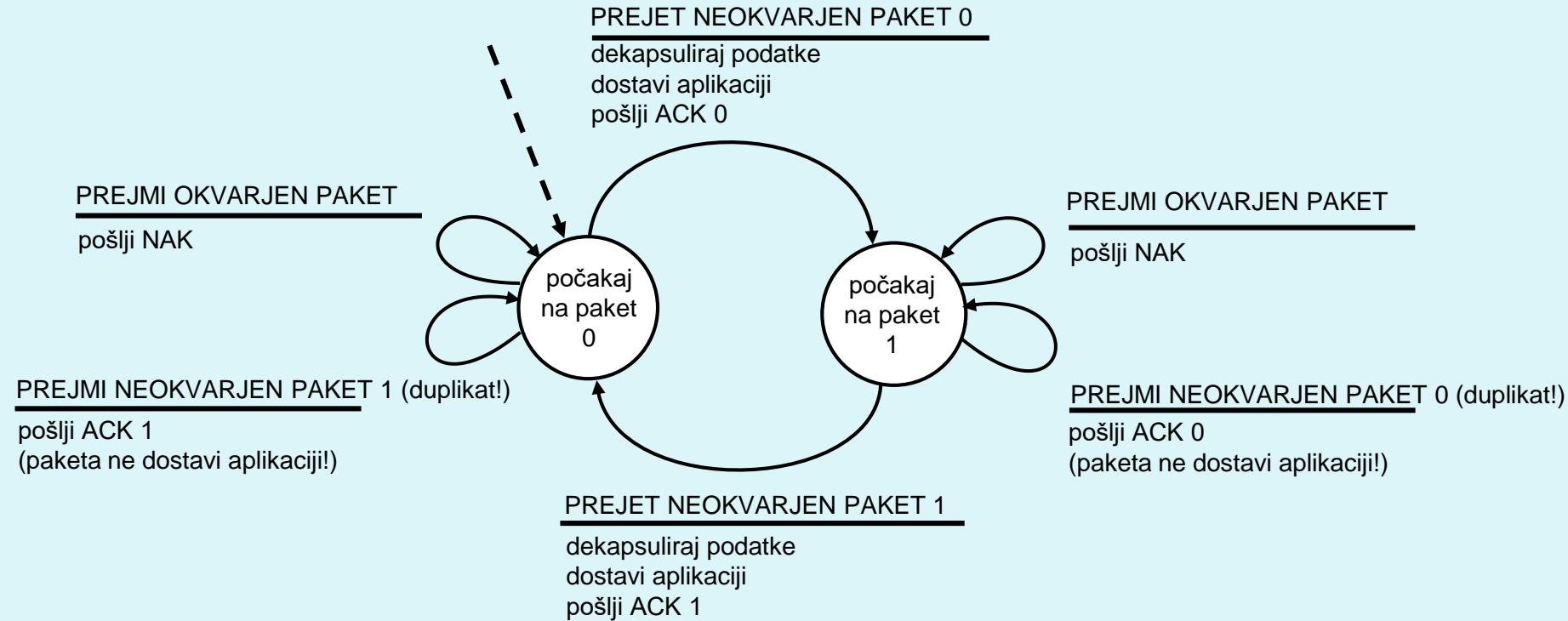


### 3. Izguba paketov ali potrditev

- 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,

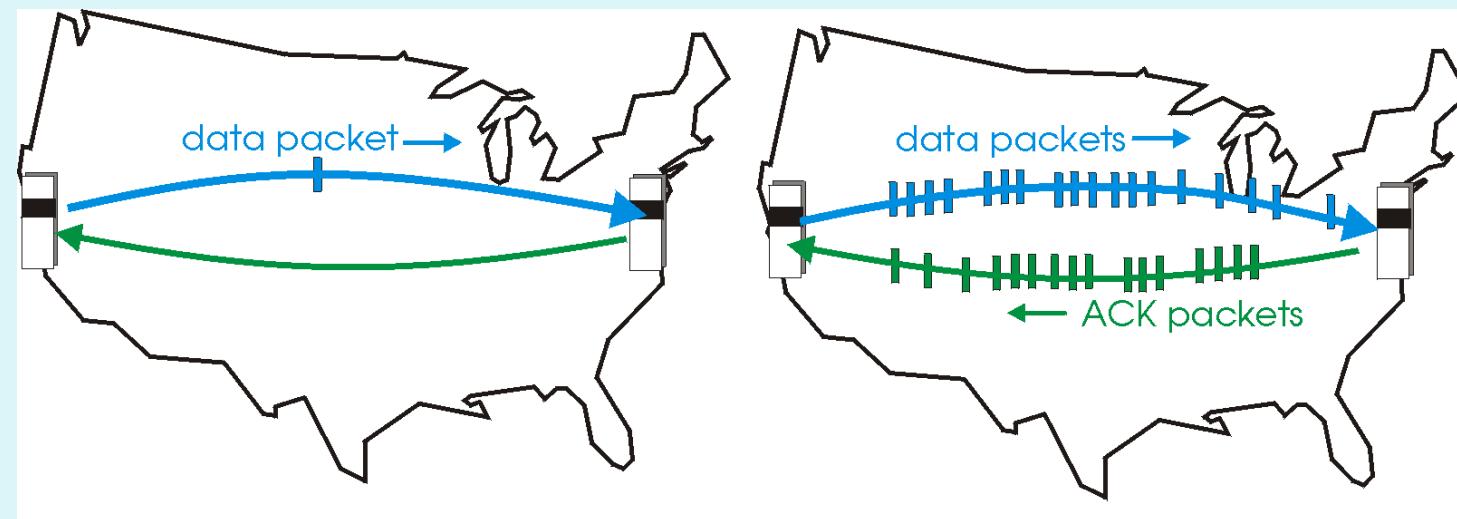
**NEPOSREDNO POTRJEVANJE: uporaba ACK in NAK**

**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**



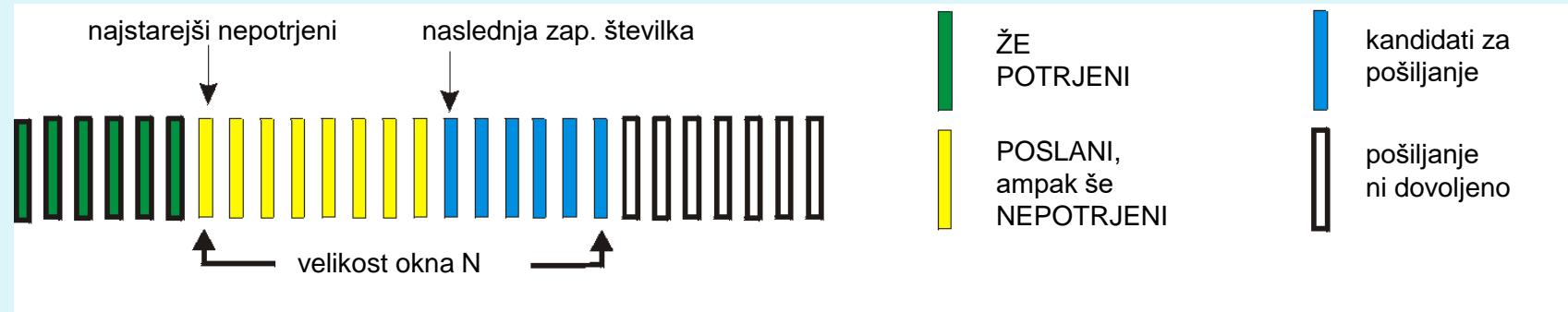
- ker lahko istočasno potuje več paketov, brez sprotnega čakanja na njihovo potrditev, je izkoriščenost kanala večja

## 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*)

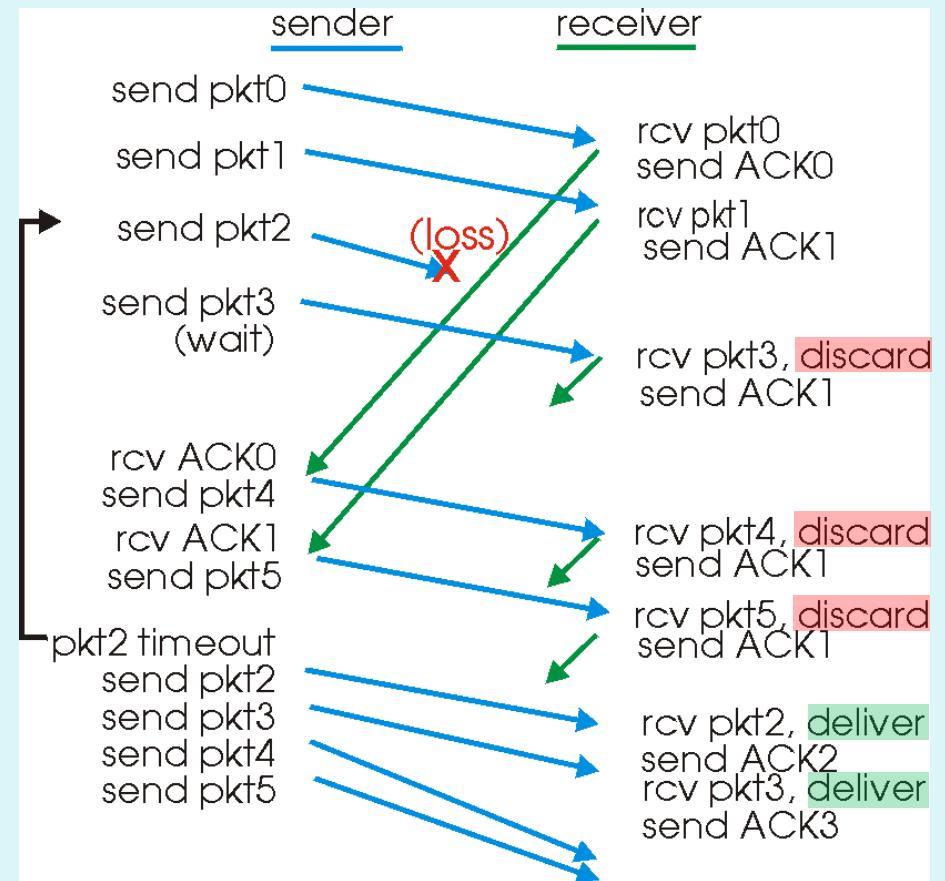
## 5. Tekoče pošiljanje: ponavljanje N nepotrjenih

- 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](#)



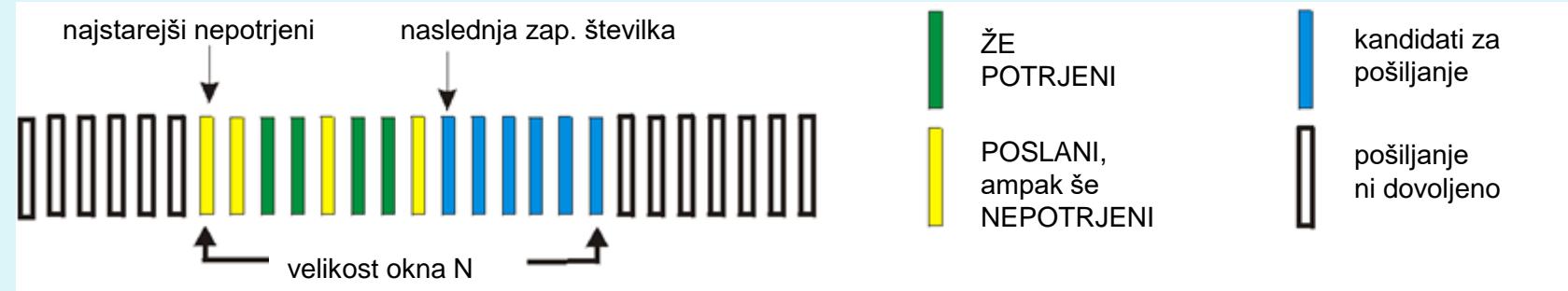
## 5. Tekoče pošiljanje: ponavljanje N nepotrjenih

- 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 poslati ponovno, ko poteče štoparica za najstarejšega

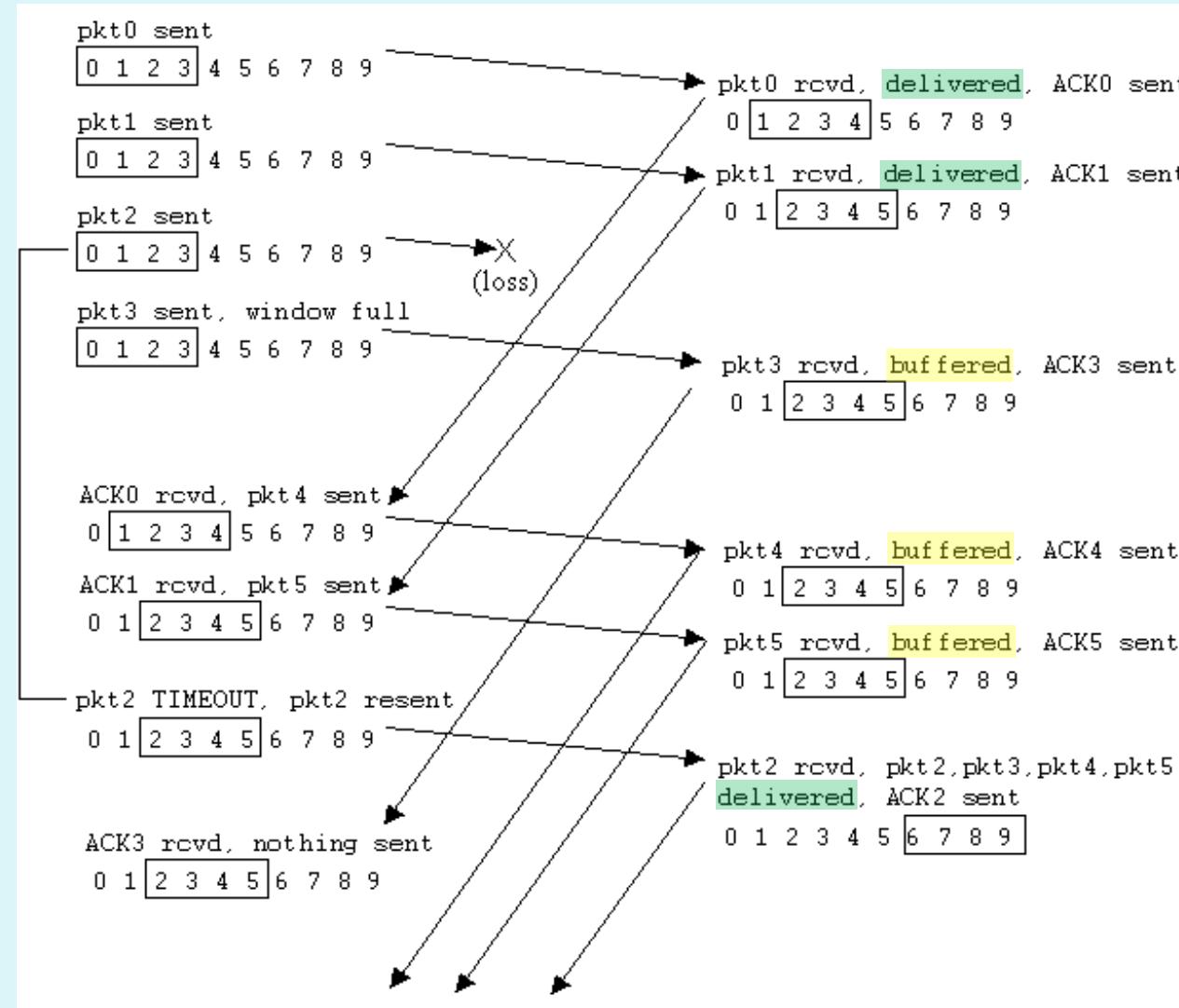


## 5. Tekoče pošiljanje: ponavljanje izbranih

- 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



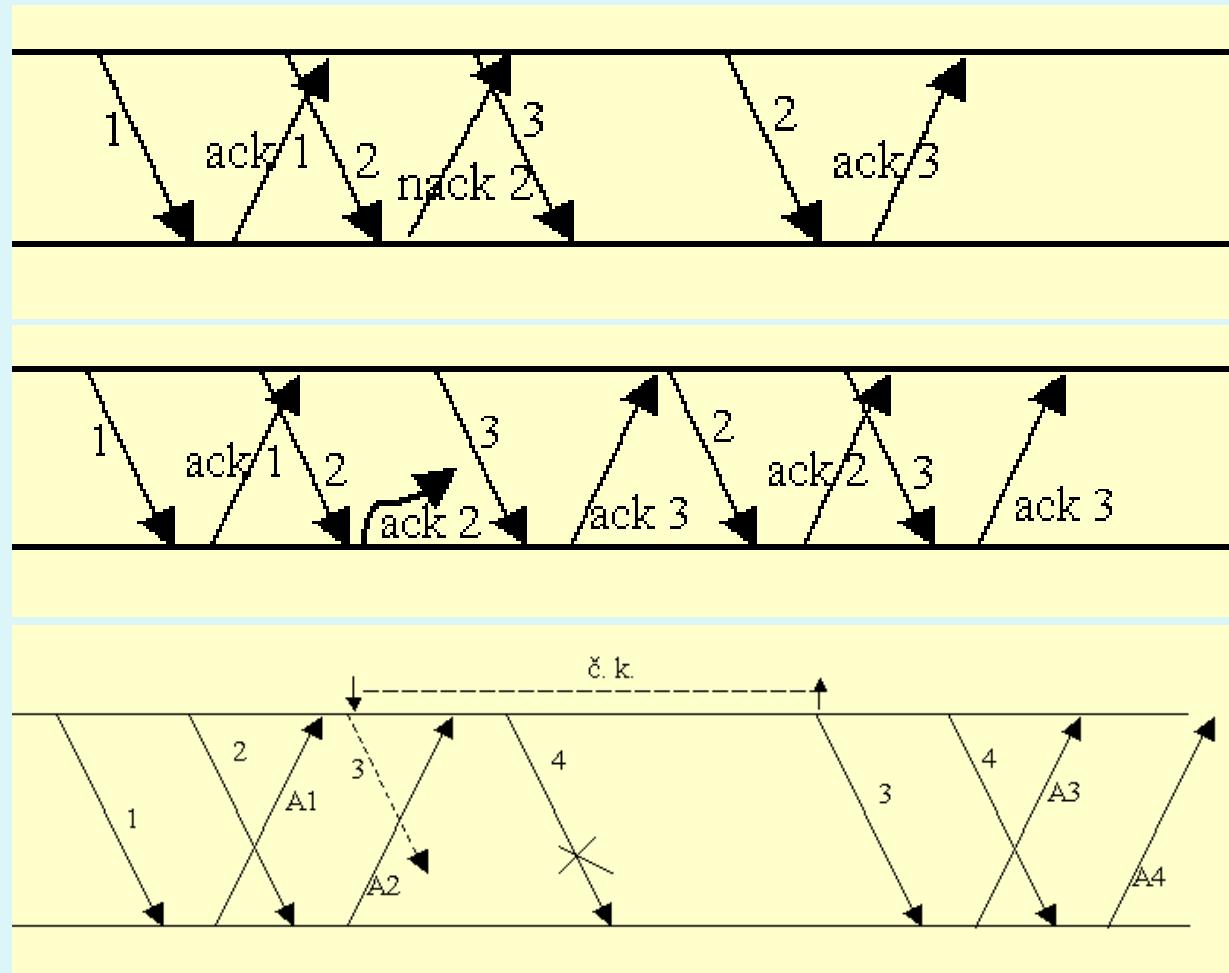
# Potrjevanje

- 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?



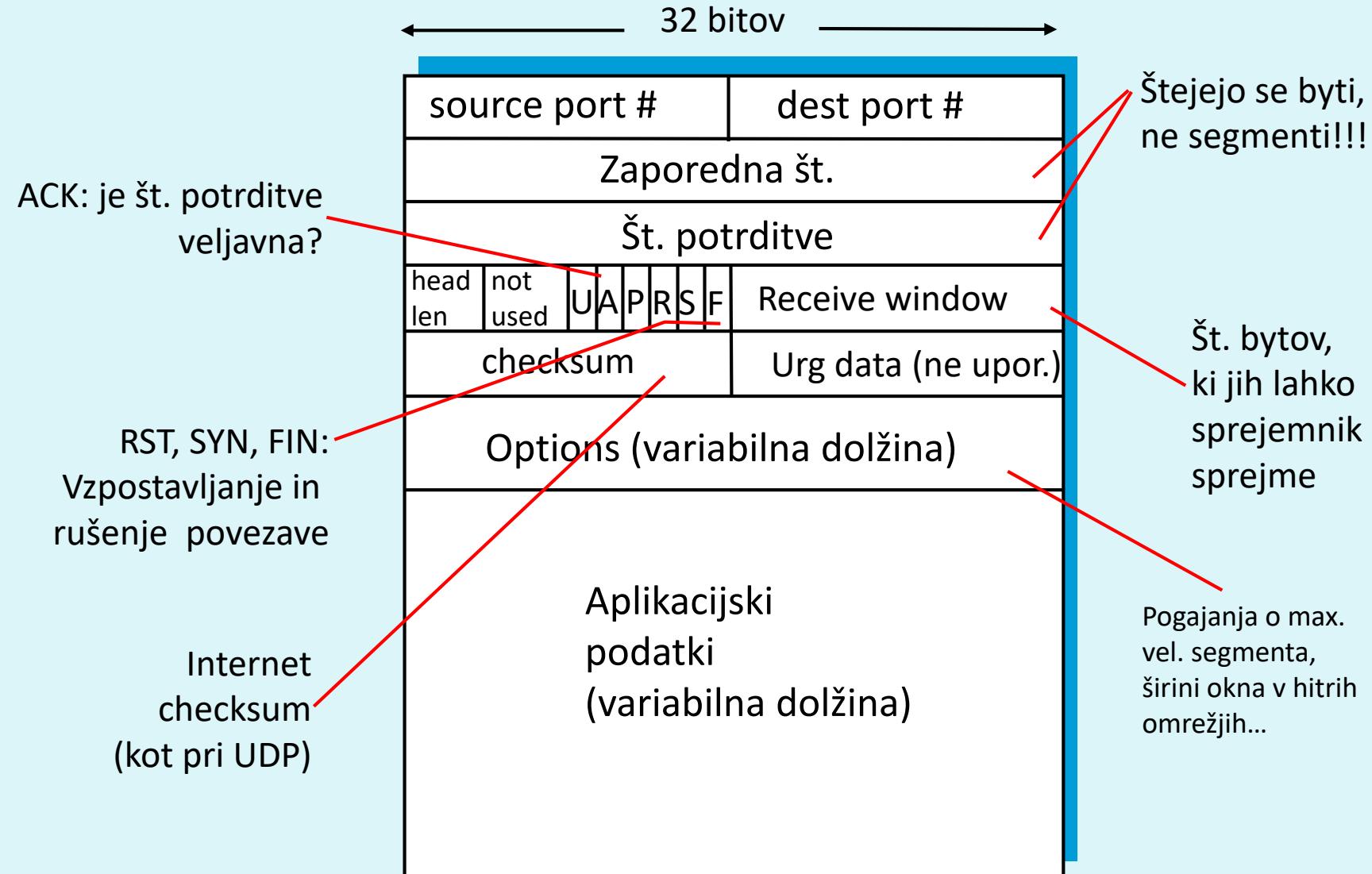
## Vaja 2

- 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?

# Konstruirani osnutek protokola TCP: ponovitev

- pošiljanje in prejemanje podatkov
  - tekoče pošiljanje namesto sprotnega
- reševanje napak pri prenosu (detekcija napak in potrjevanje z ACK in NAK)
  - poenostavljeno potrjevanje (samo pozitivne potrditve ACK)
  - optimizirano potrjevanje (kumulativno)
- obravnava izgubljenih paketov (ponovno pošiljanje)
  - odpornost na izgubljene potrditve paketov
  - obravnava duplikatov

# TCP segment: naslednjič



# Naslednjič gremo naprej!

- protokol TCP

