

# Glava 3: Nivo transporta

## Ciljevi:

- Shvatiti principe na kojima počivaju servisi nivoa transporta:
  - Multipleksiranje/demultipleksiranje
  - Pouzdan prenos podataka
  - Kontrola protoka
  - Kontrola zagušenja
- Protokoli transportnog nivoa na Internetu:
  - UDP: nekonektivni transport
  - TCP: konektivni transport
  - TCP kontrola zagušenja
  - QUIC

# Glava 3: Sadržaj

## 3.1 Servisi nivoa transporta

## 3.2 Multipleksiranje i demultipleksiranje

## 3.3 Nekonektivni transport: UDP

## 3.4 Konektivni transport: TCP

- Struktura segmenta
- Pouzdani prenos podataka
- Kontrola protoka
- Upravljanje vezom

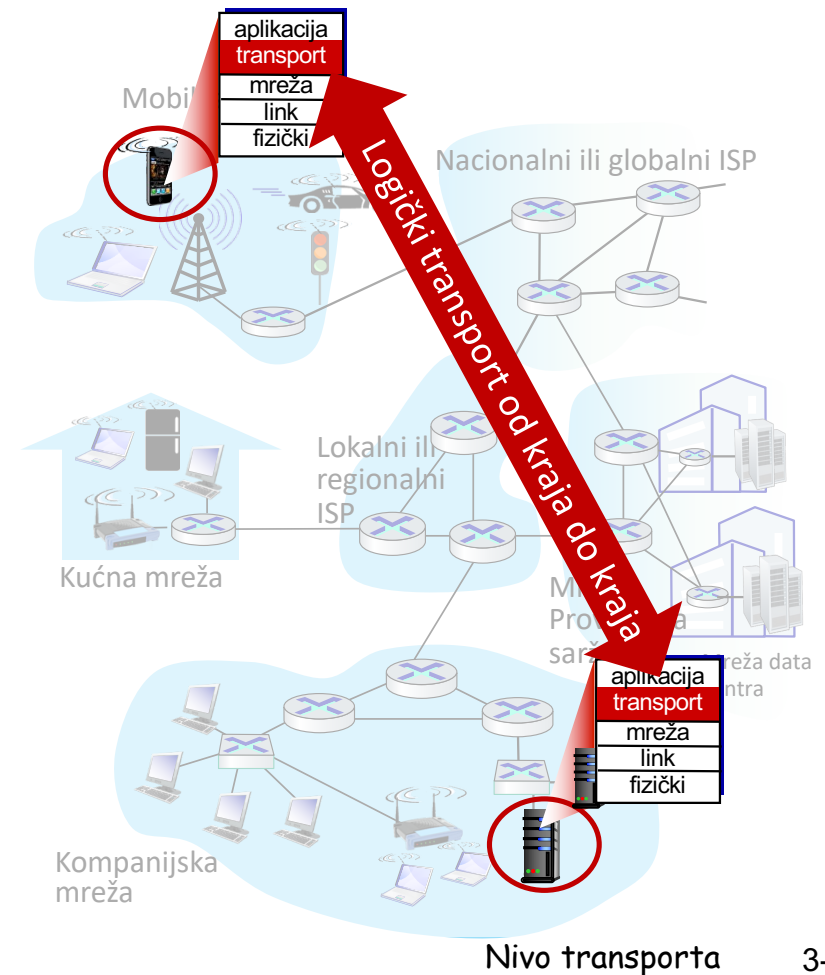
## 3.5 Principi kontrole zagušenja

## 3.6 TCP kontrola zagušenja

## 3.7 QUIC

# Transportni servisi i protokoli

- obezbjeđuju **logičku komunikaciju** između aplikacija koje se odvijaju na različitim hostovima
- transportni protokoli se implementiraju na krajnjim sistemima
  - **Predajna strana** transportnog protokola dijeli poruke u **segmente**, prosleđuje ih mrežnom nivou
  - **Prijemna strana** transportnog protokola desegmentira segmente u poruke, i prosleđuje ih nivou aplikacije
- Više od jednog transportnog protokola je na raspolaganju aplikacijama
  - Na Internetu dominiraju TCP i UDP



## Poređenje transportnog i mrežnog nivoa

- ❑ **Mrežni nivo** obezbeđuje logičku komunikaciju između hostova
- ❑ **Transportni nivo** obezbeđuje logičku komunikaciju između procesa
  - Oslanja se na servise mrežnog nivoa i poboljšava njihove osobine

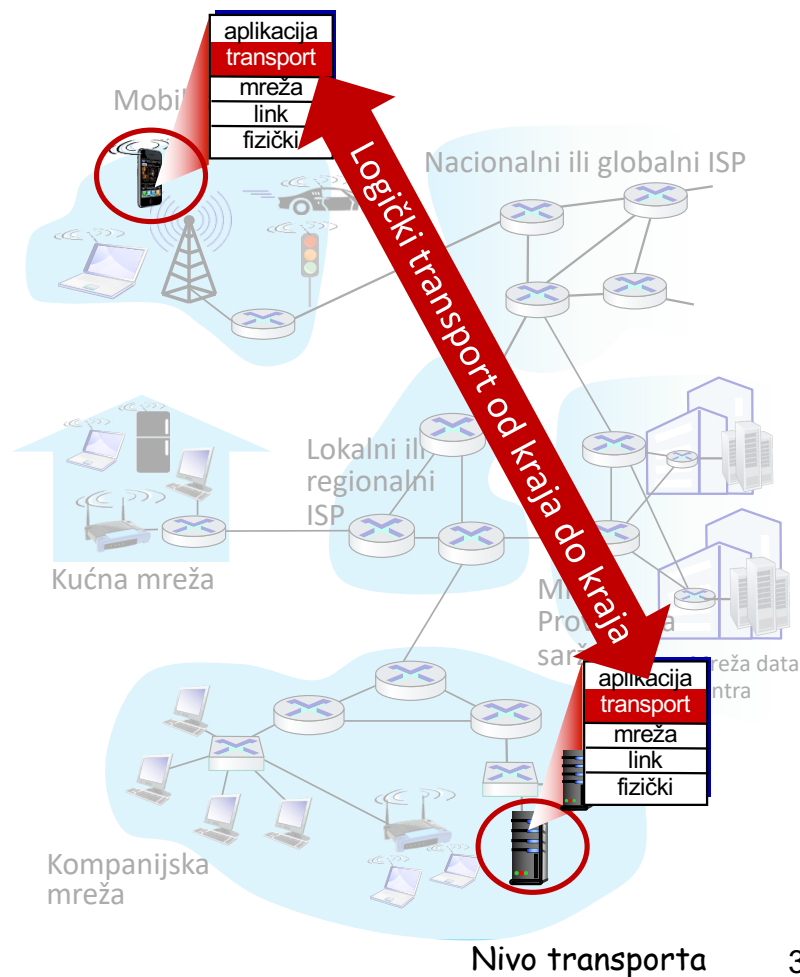
### Analogija:

*12 ljudi šalje pisma za 12 ljudi*

- ❑ procesi = ljudi
- ❑ poruke = poruke u kovertama
- ❑ hostovi = kuće u kojima ljudi žive
- ❑ transportni protokol = zapis na koverti
- ❑ mrežni protokol = poštanski servis

# Internet protokoli transportnog nivoa

- TCP
  - Multipleksiranje/demultipleksiranje saobraćaja različitih procesa
  - Pouzdana, redosledna isporuka
  - Kontrola zagušenja
  - Kontrola protoka
  - Uspostavljanje veze
- UDP
  - Multipleksiranje /demultipleksiranje saobraćaja različitih procesa
  - Detekcija greške u zaglavlju
  - nepouzdana, neredosledna isporuka
  - Bez unapređenja "best-effort" servisa IP protokola
- Servisi koji se ne pružaju:
  - Garantovano kašnjenje
  - Garantovana propusnost
  - Zaštita



# Glava 3: Sadržaj

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

3.4 Konektivni transport: TCP

- Struktura segmenta
- Pouzdani prenos podataka
- Kontrola protoka
- Upravljanje vezom

3.5 Principi kontrole zagušenja

3.6 TCP kontrola zagušenja

3.7 QUIC

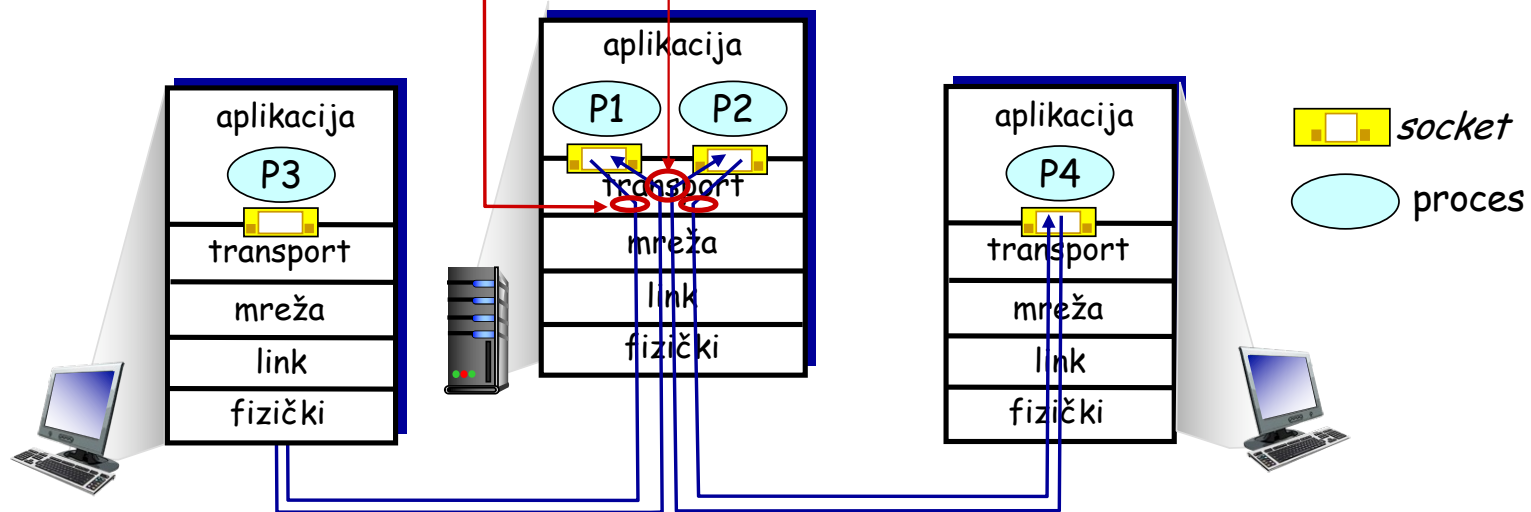
# Multipleksiranje/demultipleksiranje

## Multipleksiranje na predaji:

Manipulisanje podacima iz više *socket*-a, dodavanje transportnog zaglavlja (koristi se za demultipleksiranje)

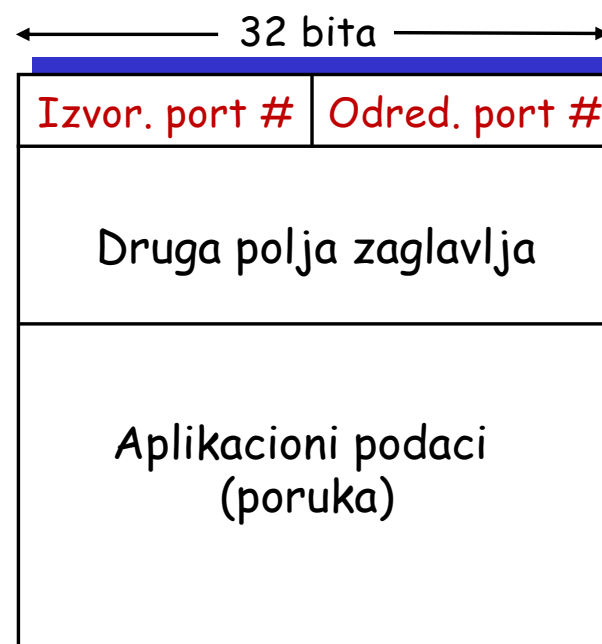
## Demultipleksiranje na prijemu:

Koristi zaglavlje za predaju primljenih segmenata pravom *socket*-u



# Kako funkcioniše demultipleksiranje?

- host prima IP datagrame
  - Svaki datagram ima izvorišnu IP adresu, odredišnu IP adresu
  - Svaki datagram nosi 1 segment nivoa transporta
  - Svaki segment ima izvorišni i odredišni broj porta
    - 16 bitni broj (0-65535)
    - 0-1023 su tzv "dobro poznati" portovi koji su unaprijed rezervisani (RFC1700, [www.iana.org](http://www.iana.org))
- host koristi IP adrese & brojeve portova da usmjeri segment na odgovarajući socket



TCP/UDP format segmenta



# Nekonektivno demultipleksiranje (UDP)

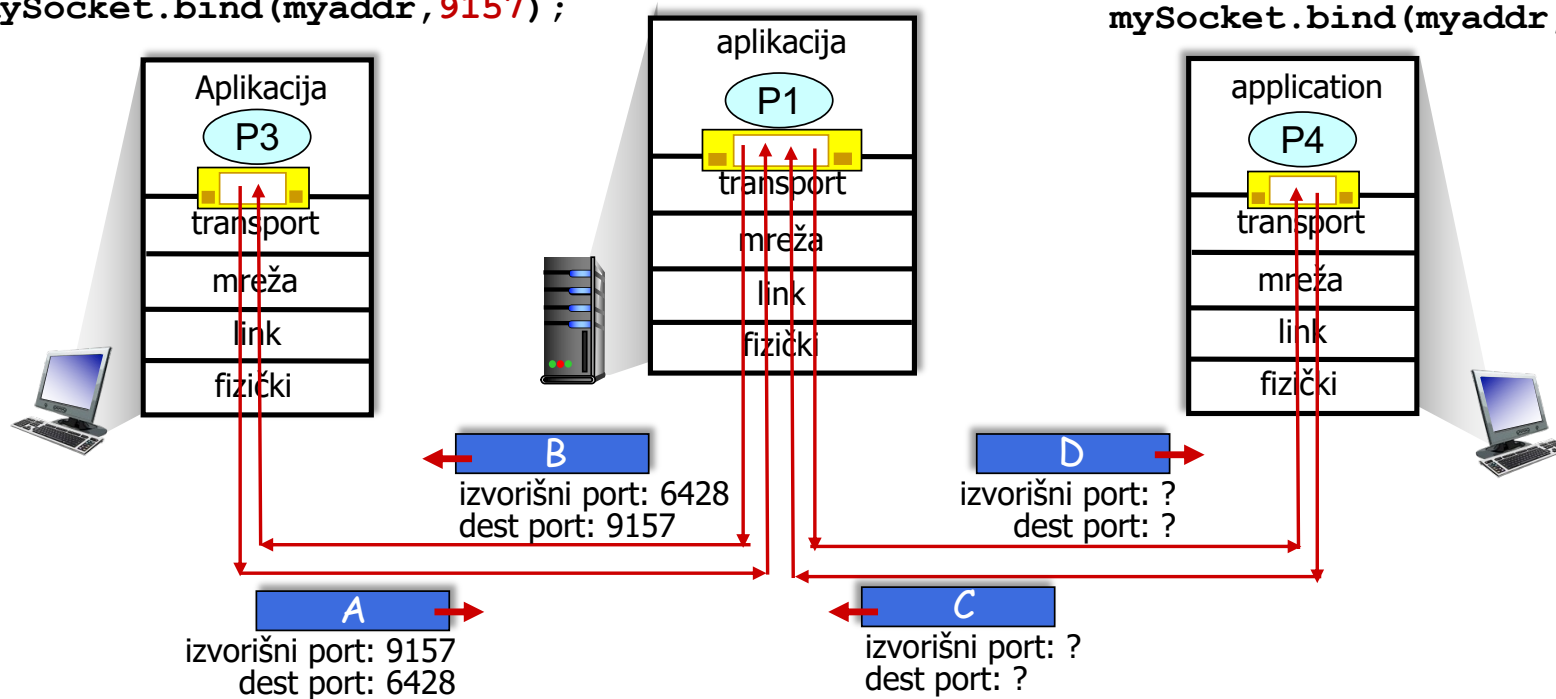
- Kada se kreira UDP *socket* transportni nivo mu odmah dodjeljuje broj porta koji ne koristi neki drugi UDP *socket* na hostu
- Klijentska strana transportnog protokola obično *socket*-u dodjeljuje ne "dobro poznate" portove 1024-65535
- UDP *socket* identifikuju dva podatka:  
(IP adresa odredišta, broj porta odredišta)
- Kada host primi UDP segment:
  - Provjerava odredišni broj porta u segmentu
  - Usmjerava UDP segment u *socket* koji ima taj broj porta
- IP datagrami sa različitim izvorišnim IP adresama i/ili izvorišnim brojevima portova se usmjeravaju na isti *socket*

# Nekonektivno multipleksiranje

```
mySocket =  
socket(AF_INET, SOCK_DGRAM)  
mySocket.bind(myaddr, 6428);
```

```
mySocket =  
socket(AF_INET, SOCK_STREAM)  
mySocket.bind(myaddr, 9157);
```

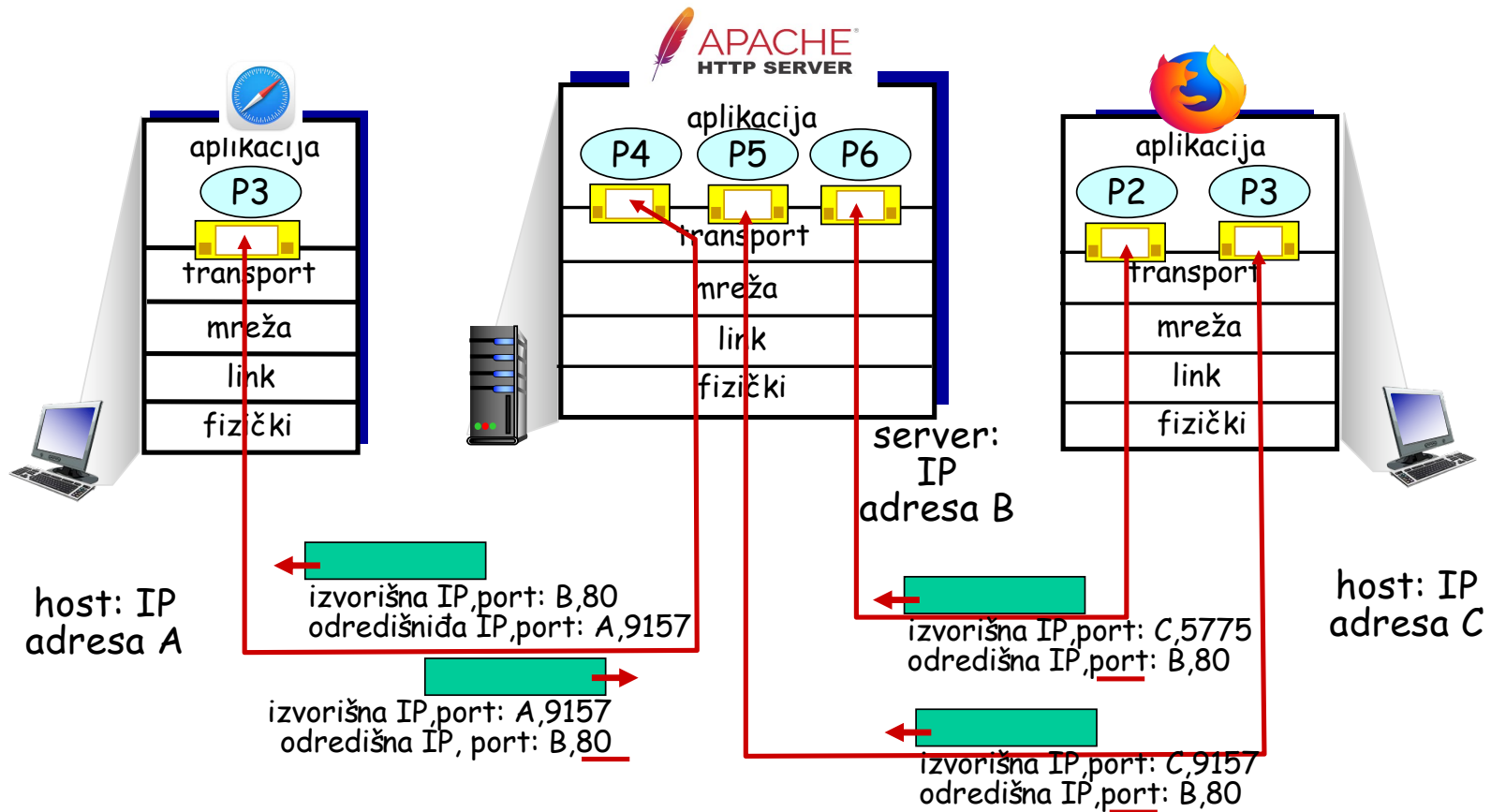
```
mySocket =  
socket(AF_INET, SOCK_STREAM)  
mySocket.bind(myaddr, 5775);
```



# Konektivno demultipleksiranje

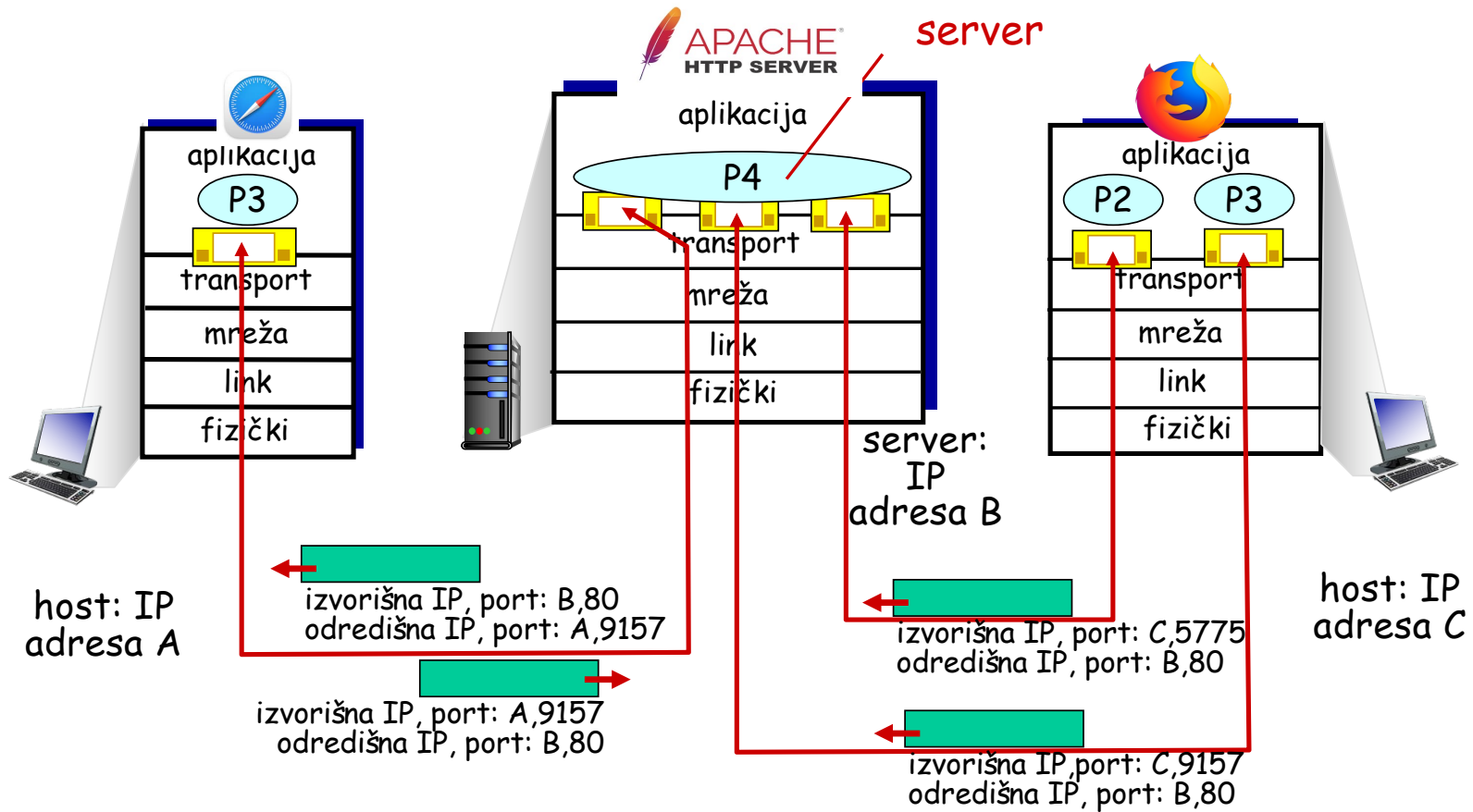
- TCP *socket* identifikuju 4 parametra:
  - Izvorišna IP adresa
  - Izvorišni broj porta
  - Odredišna IP adresa
  - Odredišni broj porta
- Prijemni host koristi sve četiri vrijednosti za usmjeravanje segmenta na odgovarajući socket
- Server host može podržavati više simultanih TCP *socket*-a:
  - svaki *socket* je identifikovan sa svoja 4 parametra
- Web serveri imaju različite *socket*-e za svakog povezanog klijenta
  - ne-perzistentni HTTP će imati različite *socket*-e za svaki zahtjev

# Konektivno demultipleksiranje



tri segmenta, adresirana na IP adresu: B,  
dest port: 80 se demultipleksiraju na različite sokete

# Konektivno demultipleksiranje



# Glava 3: Sadržaj

---

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

3.4 Konektivni transport: TCP

- Struktura segmenta
- Pouzdani prenos podataka
- Kontrola protoka
- Upravljanje vezom

3.5 Principi kontrole zagušenja

3.6 TCP kontrola zagušenja

3.7 QUIC

# UDP: User Datagram Protocol [RFC 768]

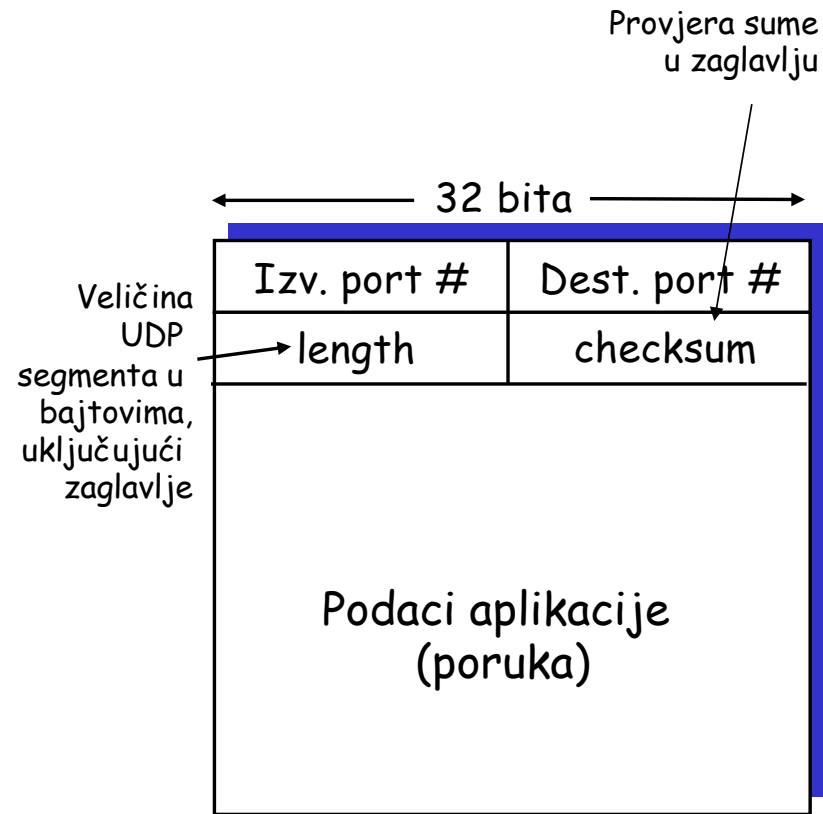
- ❑ Nema poboljšanja koja se nude Internet protokolu
- ❑ "best effort" servis, UDP segmenti mogu biti:
  - izgubljeni
  - neredosledno predati
- ❑ **nekonektivni:**
  - nema uspostavljanja veze (*handshaking*) između UDP pošiljaoca i prijemnika
  - svaki UDP segment se tretira odvojeno od drugih

## Zašto onda UDP?

- ❑ Nema uspostavljanja veze (koja povećava kašnjenje)
- ❑ Jednostavniji jer ne vodi se računa o stanju veze
- ❑ Manje zaglavlje segmenta (8B u odnosu na 20B kod TCP-a)
- ❑ Nema kontrole zagušenja tako da može slati podatke onom brzinom kojom to aplikacija želi

# UDP

- Često se koristi za "streaming" multimedijalne aplikacije
  - Tolerantne u odnosu na gubitke
  - Osjetljive na brzinu prenosa
- Drugi UDP korisnici
  - Streaming aplikacije
  - DNS
  - SNMP (zbog toga što mrežne menadžment aplikacije funkcionišu kada je mreža u kritičnom stanju)
  - HTTP/3
- Za pouzdani prenos preko UDP mora se dodati pouzdanost na nivou aplikacije
  - Oporavak od greške na nivou aplikacije
  - Kontrola zagušenja na nivou aplikacije



Format UDP segmenta  
RFC 768



# UDP

INTERNET STANDARD

RFC 768

J. Postel  
ISI  
28 August 1980

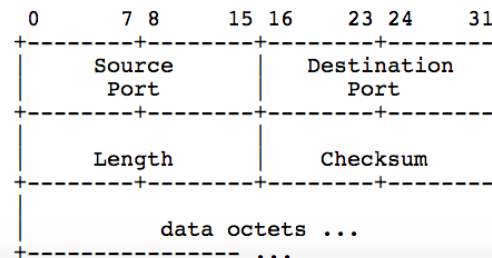
## User Datagram Protocol

### Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

### Format



# Glava 3: Sadržaj

---

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

3.4 Konektivni transport: TCP

- Struktura segmenta
- Pouzdani prenos podataka
- Kontrola protoka
- Upravljanje vezom

3.5 Principi kontrole zagušenja

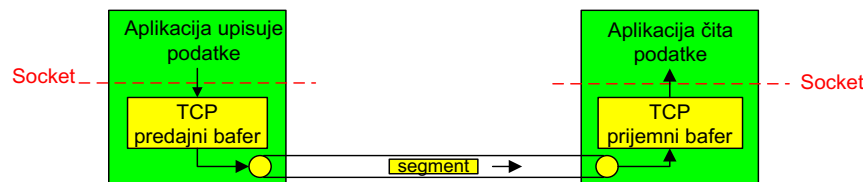
3.6 TCP kontrola zagušenja

3.7 QUIC

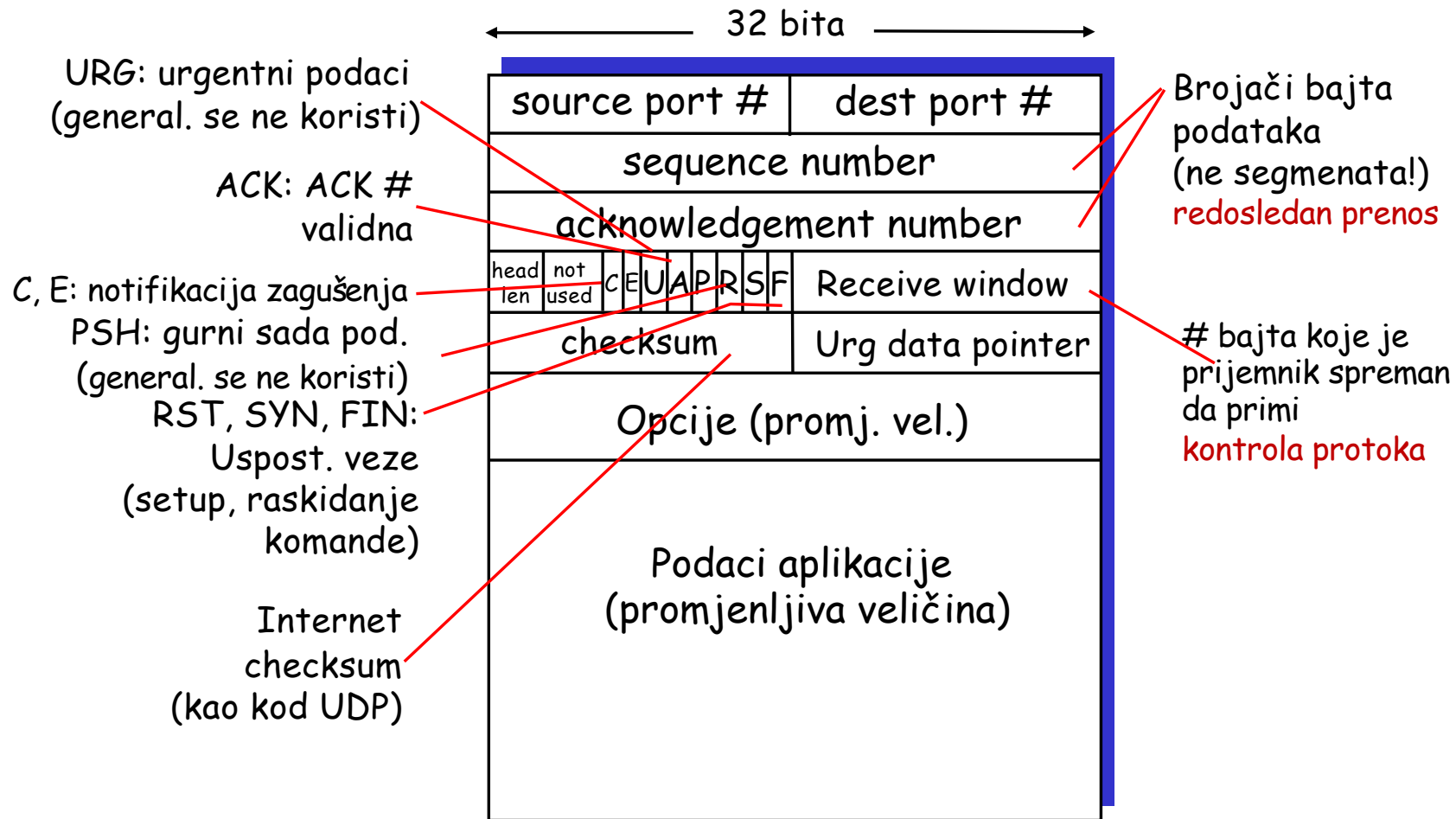
# TCP: Pregled

RFC-ovi: 793, 1122, 1323, 2018, 2581

- ❑ Tačka-tačka:
    - Jedan pošilj, jedan prij.
  - ❑ Pouzdan, redosledan prenos bajta:
    - Nema "granica poruka"
  - ❑ *Pipelined* prenos:
    - TCP kontrola zagušenja i protoka podešava veličinu prozora
  - ❑ Baferi za slanje & prijem
- ❑ *Full duplex* prenos:
    - U istoj vezi prenos u dva smjera
    - MSS: maksimalna veličina podataka sloja aplikacije u segmentu (1460B, 536B, 512B)
  - ❑ Konektivan:
    - *Handshaking* (razmjena kontrolnih poruka) inicira pošiljalac prije slanja poruka protokola nivoa aplikacije
  - ❑ Kontrola protoka:
    - Pošiljalac ne može "zagušiti" prijemnika



# TCP struktura segmenta (21B-1480B)



# TCP brojevi u sekvenci, ACK-ovi

## Brojevi u sekvenci:

- Dodjeljuje se broj prvom bajtu iz sadržaja segmenta
- Inicijalne vrijednosti se utvrđuju na slučajan način.

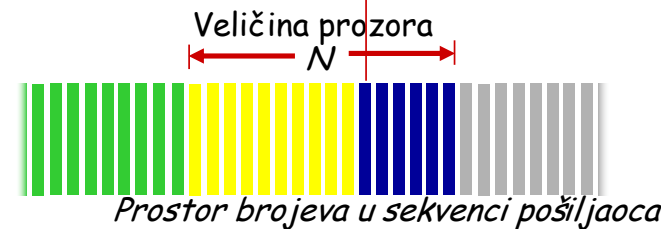
## Potvrde (ACK):

- Broj sekvence sledećeg bajta koji se očekuje sa druge strane
- Kumulativni ACK

**Q:** Kako se prijemnik ponaša prema *out-of-order* segmentima?

## Odlazni segment pošiljaoca

Source port #	dest port #
sequence number	
acknowledgement	
number	rwnd
checksum	urg pointer



Poslate potvrde

poslati, nepotvrđeni ("in-flight")

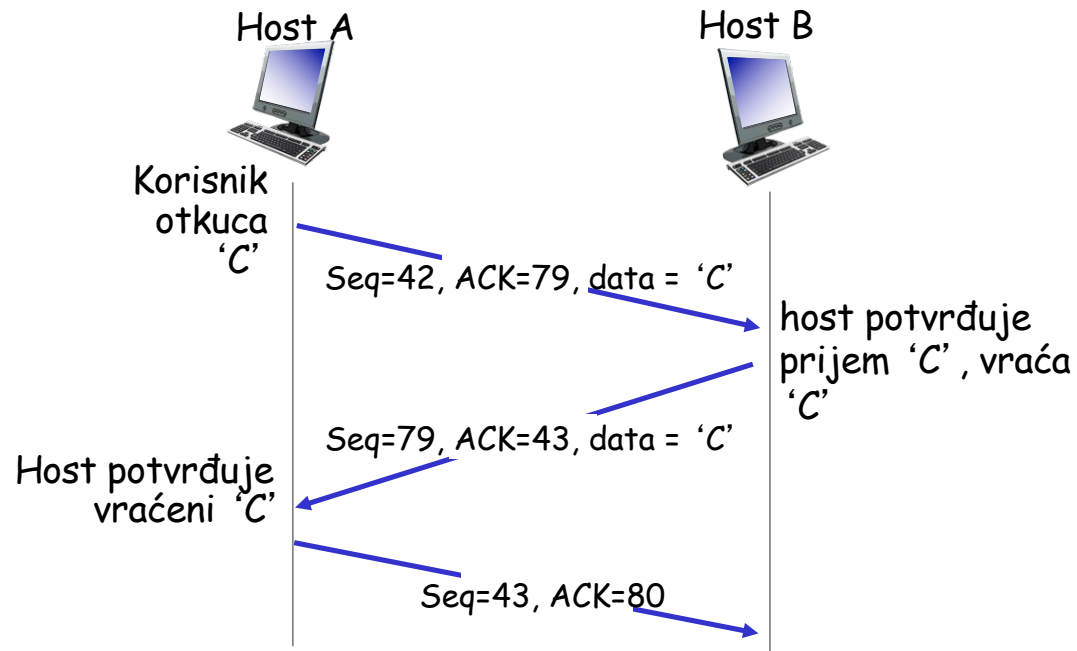
mogu se koristiti

neiskorišćeni

## Segment koji dolazi pošiljaocu

source port #	dest port #
sequence number	
acknowledgement	
	A
checksum	urg pointer

# TCP brojevi u sekvenci, potvrde



Jednostavni Telnet scenario

# TCP pouzdani prenos podataka

- TCP kreira pouzdani servis baziran na npouzdanom IP servisu
- *Pipelined* segmenti
- Kumulativne potvrde
- TCP koristi jedan tajmer za pokretanje retransmisije
- Retransmisije su triggerovane sa:
  - *timeout* događajima
  - duplim ack-ovima
- Na početku treba razmotriti pojednostavljenu TCP predajnu stranu:
  - Ignorišu se duplirani ack-ovi
  - Ignorišu se kontrole protoka i zagušenja

# Događaji vezani za TCP pošiljaoca

## 1. Podaci primljeni od aplikacije:

- ❑ Kreiranje segmenta sa brojem u sekvenci
- ❑ Broj u sekvenci je *byte-stream* broj prvog bajta podataka u segmentu
- ❑ Startuje se tajmer ako to već nije urađeno
- ❑ Interval *timeout-a* se izračunava po odgovarajućoj formuli

## 2. timeout:

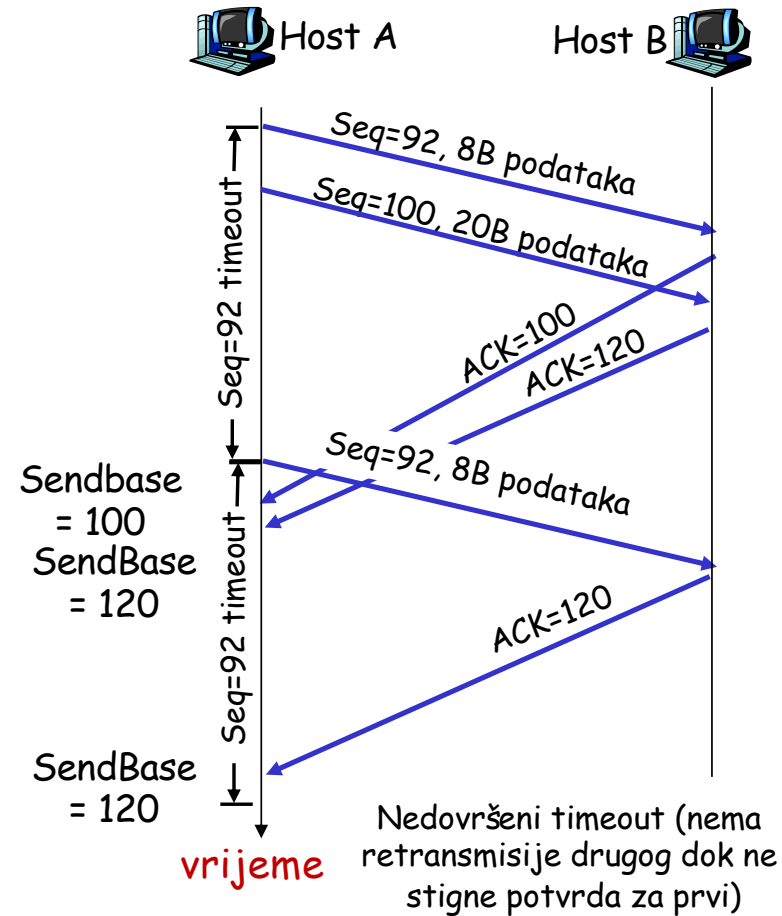
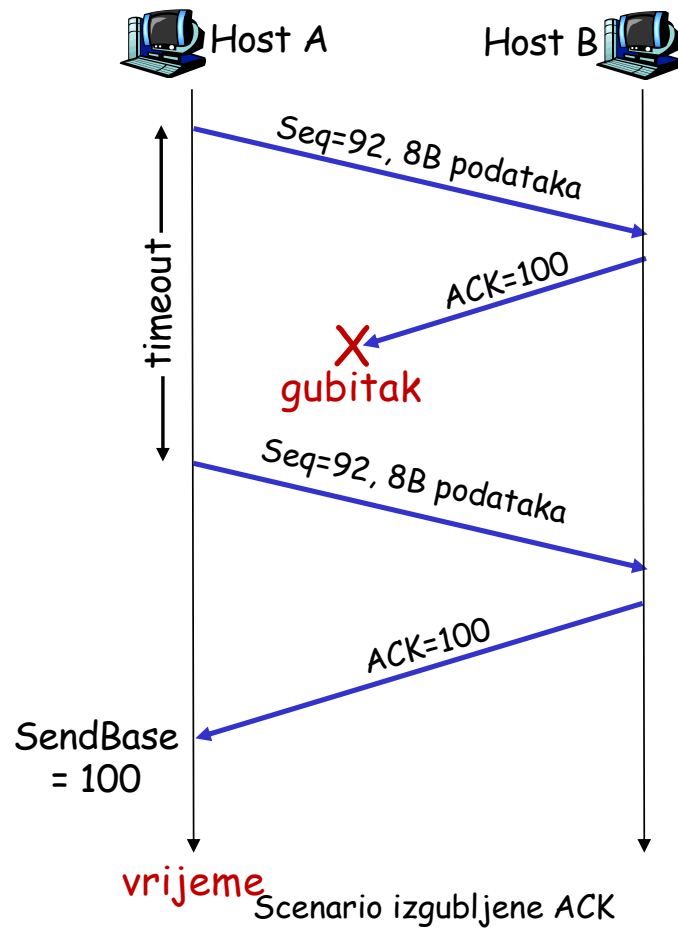
- ❑ Ponovo se šalje segment koji je izazvao timeout
- ❑ restartovati tajmer

## 3. Ack primljen:

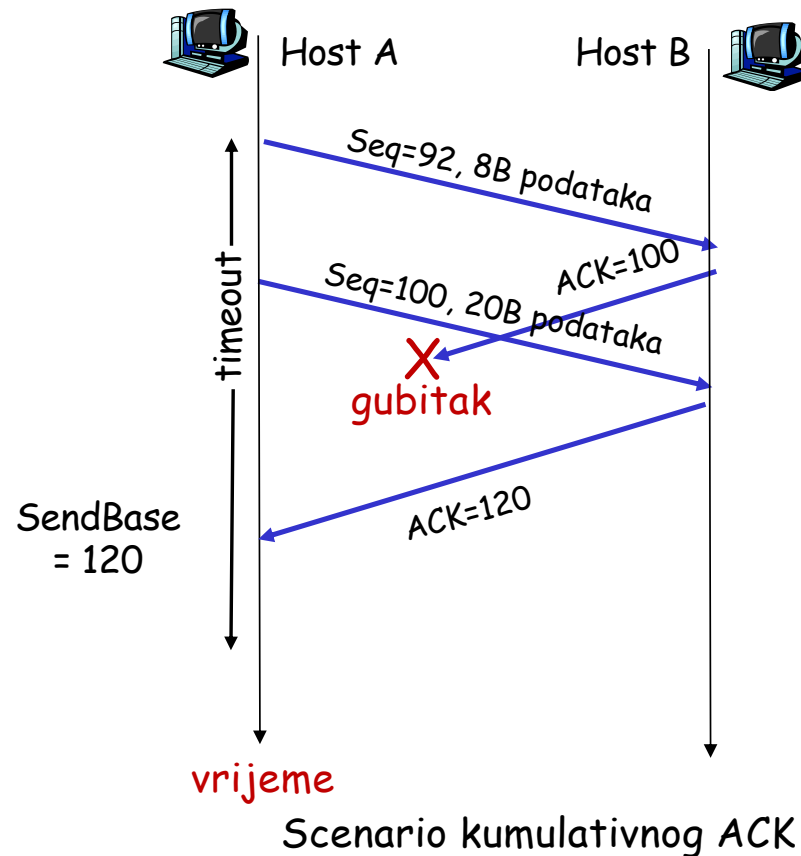
- ❑ Ako se potvrdi prijem ranije nepotvrđenog segmenta
  - Pravi se odgovarajući *update*
  - Startuje se tajmer ako postoje segmenti koji čekaju



# TCP: scenariji retransmisije



# TCP: scenariji retransmisije



U slučaju kada istekne *timeout* period, TCP se više ne pridržava formule za izračunavanje *timeout* intervala o kojoj će biti riječi. Umjesto nje TCP duplira raniju vrijednost *timeout* intervala.

# TCP Round Trip Time i Timeout

P: kako postaviti TCP vrijeme timeout-a?

- Duže od RTT-a
  - ali RTT varira
- Suviše kratko izaziva prerani timeout izaziva nepotrebne retransmisije
- Previše dugo ima za posledicu sporu reakciju na gubitak segmenta
- Potrebna je aproksimacija RTT-a

P: kako aproksimirati RTT?

- **SampleRTT**: mjeriti vrijeme od slanja segmenta do prijema ACK
  - Ignorirati retransmisije
  - Radi se za samo jedan nepotvrđeni segment
- **SampleRTT** će varirati, želja je za što boljom estimacijom RTT
  - Više mjerenja, a ne samo trenutno SampleRTT

P: Da li SampleRTT vezivati za svaki nepotvrđeni segment?

P: Zašto ignorirati retransmisije?

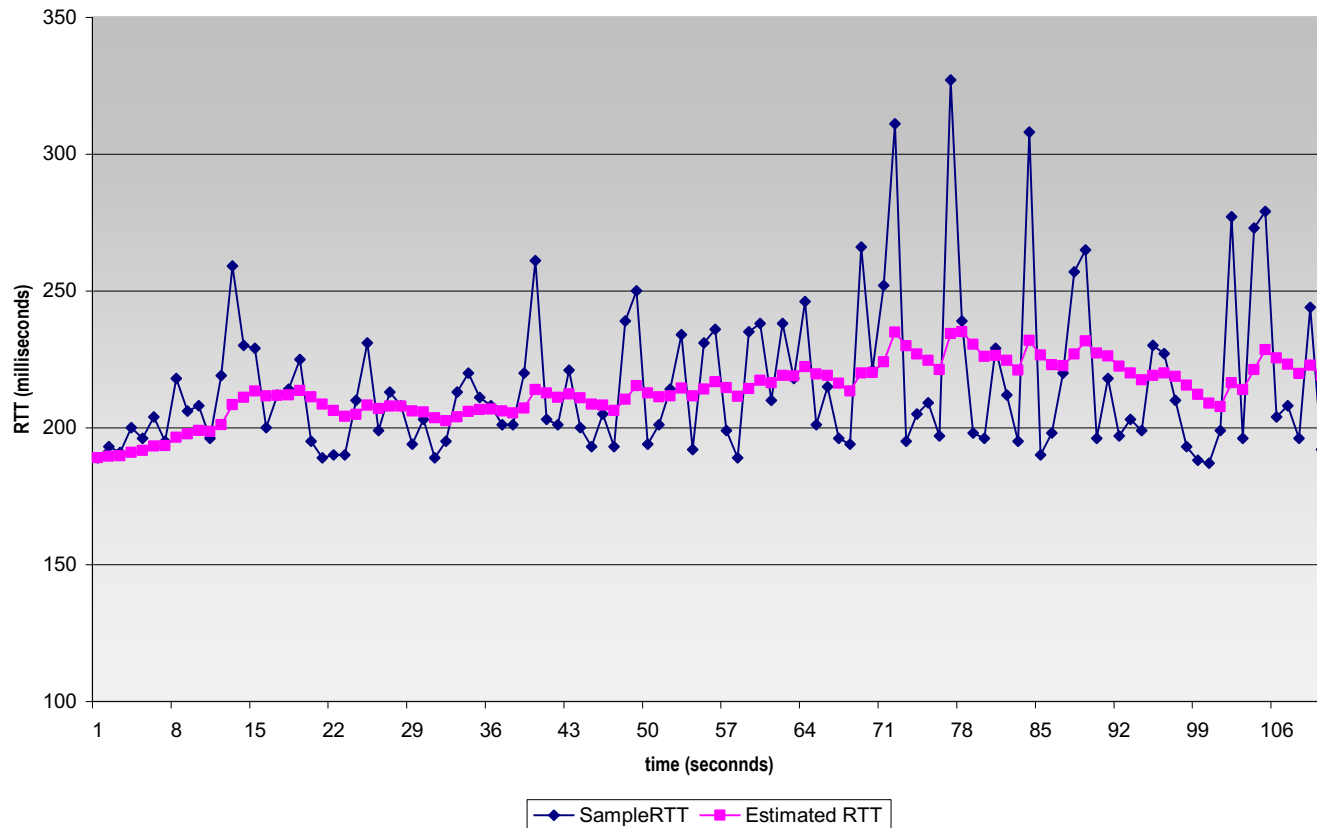
# TCP Round Trip Time and Timeout

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- ❑ Uticaj prošlosti opada po eksponencijalnoj raspodjeli
- ❑ *Exponential weighted moving average* (EWMA) ili eksponencijalno ponderisani klizni prosjek
- ❑ Tipična vrijednost:  $\alpha = 0.125$

# Primjer RTT estimacije:

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr



# TCP Round Trip Time i Timeout

## Setovanje timeout-a

- EstimatedRTT + "sigurnosna margina"
  - Velika varijacija u EstimatedRTT -> velika sigurnosna margina

- Prvo se estimira koliko SampleRTT odstupa od EstimatedRTT:

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(tipično,  $\beta = 0.25$ )

← EWMA od ove razlike

Timeout interval za izračunava po formuli:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

## TCP generisanje ACK [RFC 5681]

Događaj na prijemu	TCP akcije prijemnika
Dolazak in-order segmenta sa očekivanim brojem u sekvenci. Svi podaci do očekivanog broja su potvrđeni	ACK sa kašnjenjem. Čeka do 500ms za sledeći segment. Ako nema sledećeg, šalje ACK.
Dolazak in-order segmenta sa očekivanim brojem u sekvenci. Potvrđivanje prijema drugog segmenta u toku.	Odmah šalje jednu kumulativnu ACK, potvrđujući oba in-order segmenta.
Dolazak out-of-order segmenta sa većim brojem u sekvenci od očekivanog. Detekcija prekida.	Odmah šalje duplikat ACK, indicirajući broj u sekvenci očekivanog bajta.
Dolazak segmenta koji djelimično ili potpuno popunjava prekid.	Odmah šalje ACK, omogućavajući da segment popuni prekid.

# "Fast Retransmit"

---

- *Time out period* je često predugačak:
  - Dugo kašnjenje prije retransmisije izgubljenog paketa
- Detekcija izgubljenog segmenta preko dupliranih ACK-ova.
  - Pošiljalac često šalje mnogo segmenata
  - Ako je segment izgubljen, najvjerovatnije će biti dosta dupliranih ACK-ova.
- Ako pošiljalac primi 3 ACK za isti segment, pretpostavlja se da je segment poslije potvrđenog izgubljen:
  - "fast retransmit": retransmisije segmenta prije nego što je tajmer istekao

Da li TCP ima GBN ili "*selective repeat*" kontrolu greške?  
Zašto 3 a ne dva ACK?



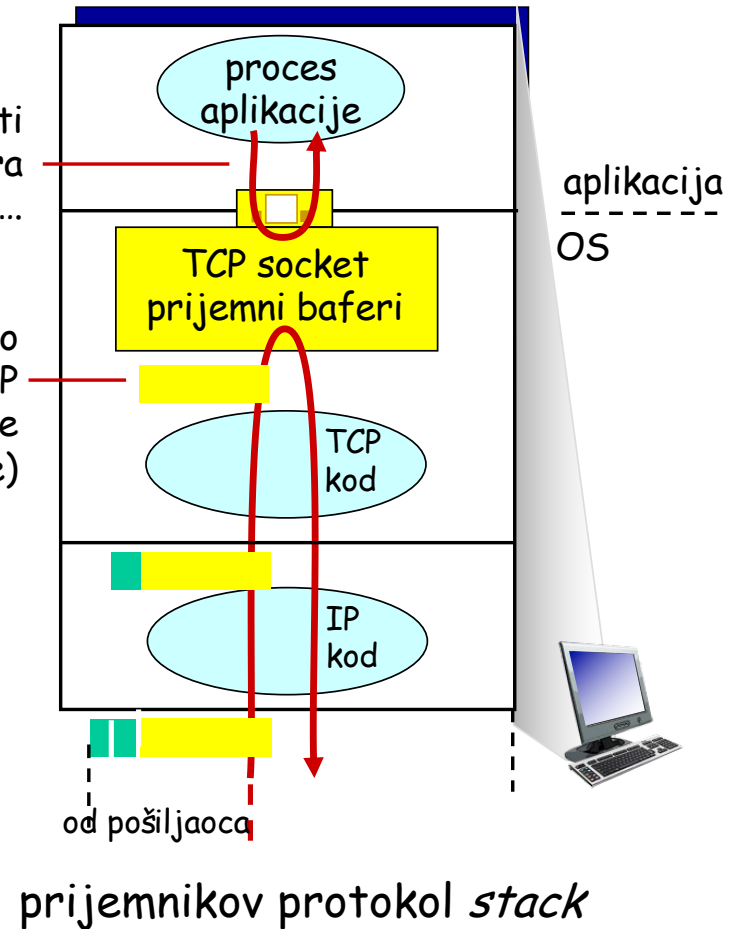
# TCP kontrola protoka

Aplikacija može ukloniti podatke iz bafera TCP socket-a ....

... sporije nego što TCP prijemnik predaje (pošiljalac šalje)

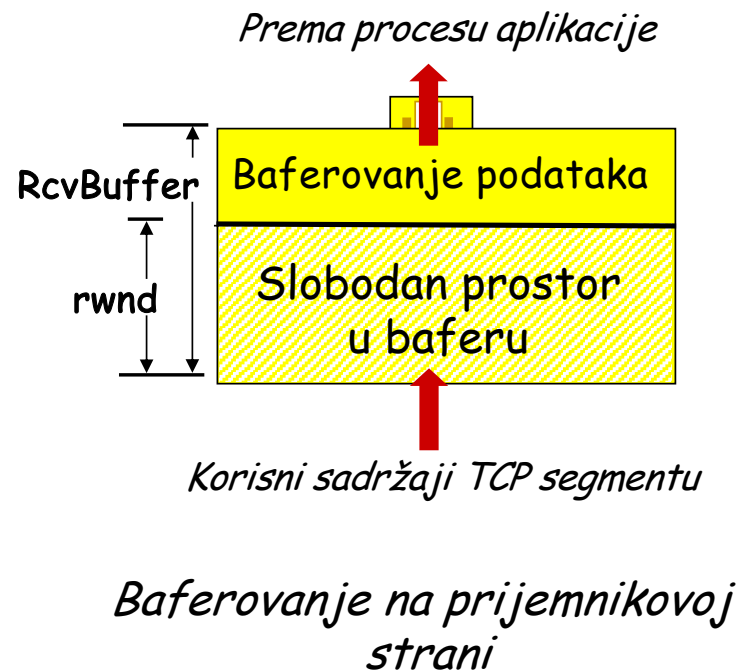
## *Kontrola protoka*

Prijemnik kontroliše pošiljaoca, tako da pošiljalac neće zagušiti prijemnikov bafer šaljući podatke velikom brzinom

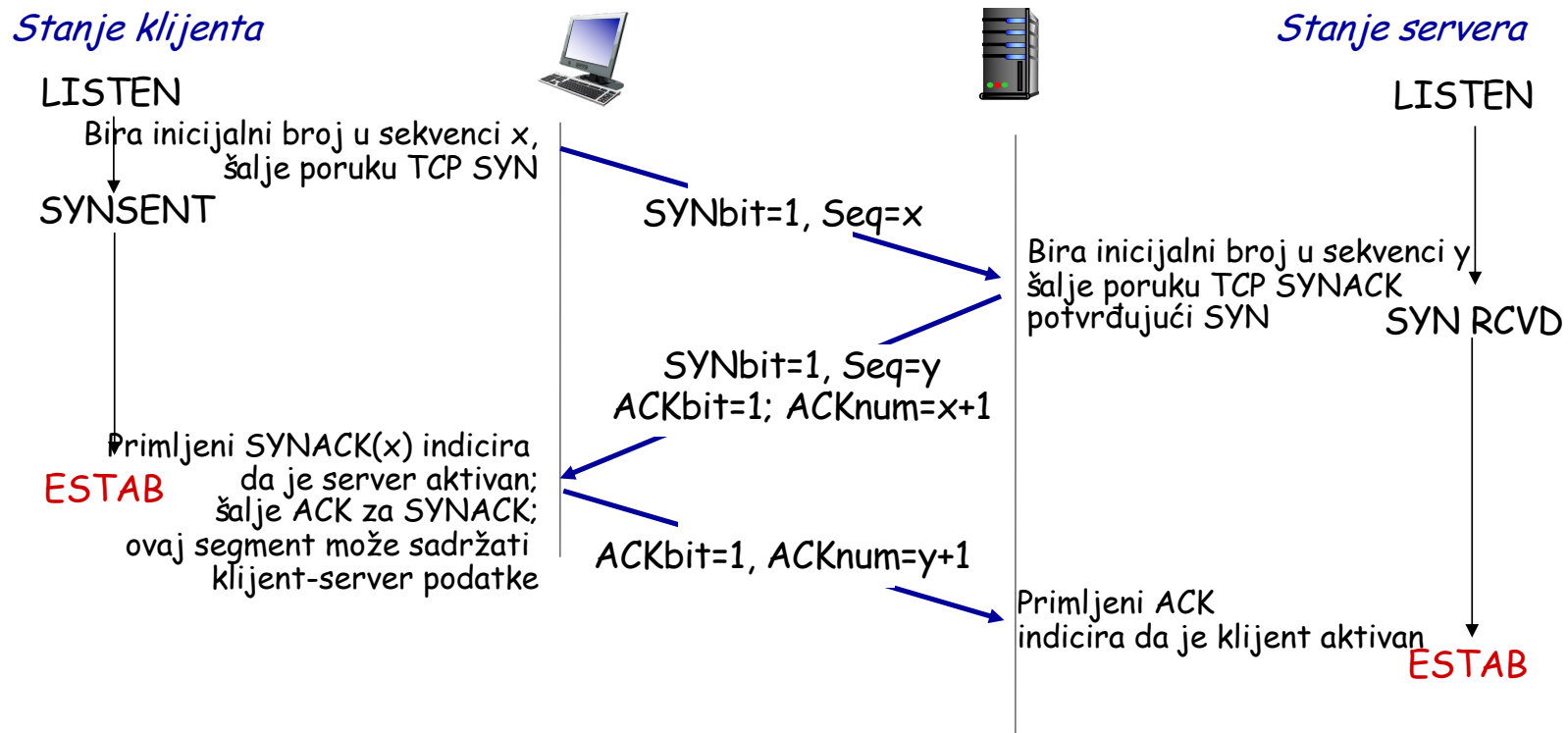


# TCP kontrola protoka

- ❑ Prijemnik oglašava slobodan prostor u baferu podešavanjem vrijednosti u polje `rwnd` u zaglavlju TCP segmenta
  - Veličina `RcvBuffer` se podešava u opcijama `socket`-a (tipična vrijednost 4096B)
  - Mnogi OS podešavaju `RcvBuffer`
- ❑ Pošiljalac ograničava broj nepotvrđenih (“in-flight”) podataka na vrijednost prijemnikovog `rwnd`
- ❑ Garantuje da se ne prepuni bafer



# TCP 3-way handshake



# TCP: zatvaranje konekcije

## Stanje klijenta

ESTAB

clientSocket.close()

FIN\_WAIT\_1

Ne može slati  
ali može  
primati

FIN\_WAIT\_2

Čeka da server  
zatvori

TIMED\_WAIT

čekanje u trajanju  
od dva vremena "života"  
segmenta

CLOSED



FINbit=1, seq=x

ACKbit=1; ACKnum=x+1

FINbit=1, seq=y

ACKbit=1; ACKnum=y+1

Još uvijek  
može slati

Ne može  
više slati

## Stanje servera

ESTAB

CLOSE\_WAIT

LAST\_ACK

CLOSED

# Glava 3: Sadržaj

---

- 3.1 Servisi nivoa transporta
- 3.2 Multipleksiranje i demultipleksiranje
- 3.3 Nekonektivni transport: UDP

## 3.4 Konektivni transport: TCP

- Struktura segmenta
- Pouzdani prenos podataka
- Kontrola protoka
- Upravljanje vezom

## 3.5 Principi kontrole zagušenja

## 3.6 TCP kontrola zagušenja

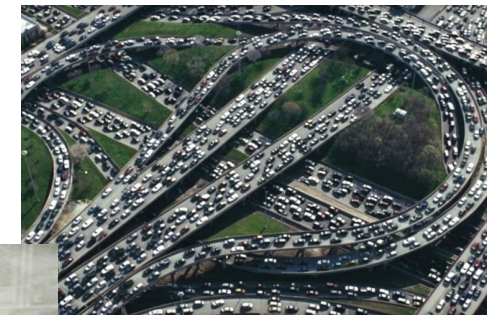
## 3.7 QUIC

# Principi kontrole zagušenja

---

## Zagušenje:

- ❑ javlja se kada korisnici generišu količinu saobraćaja koju mreža ne može da prenese
- ❑ pojavni oblici:
  - ❑ velika kašnjenja (baferovanje u ruterima)
  - ❑ gubici paketa (puni baferi u ruterima)
- ❑ razlikuje se od kontrole protoka
- ❑ Veliki problem!



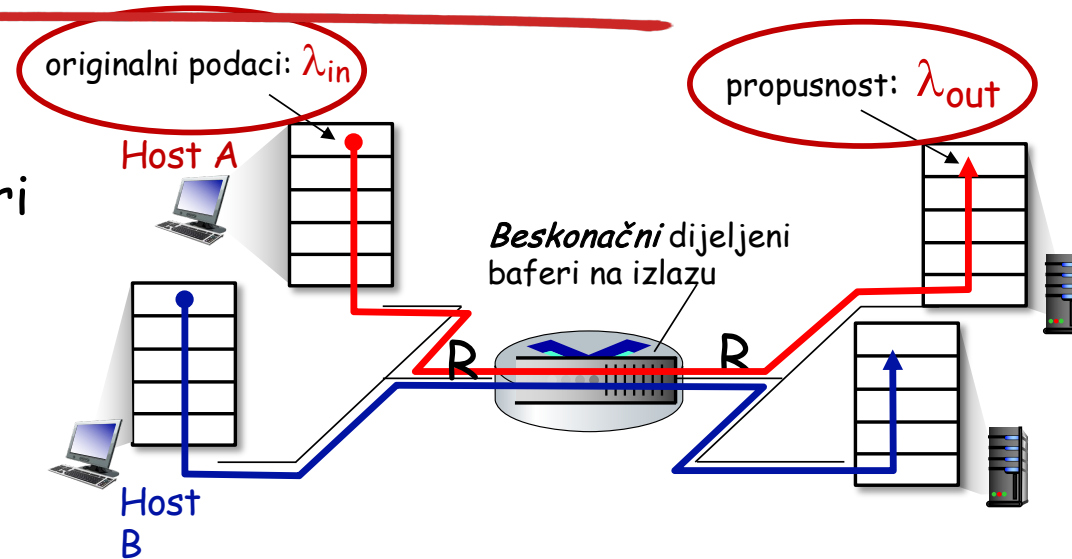
**Kontrola zagušenja:** previše pošiljaoca, šalju prevelikom brzinom

**Kontrola protoka:** pošiljalac šalje brže nego što prijemnik može da primi

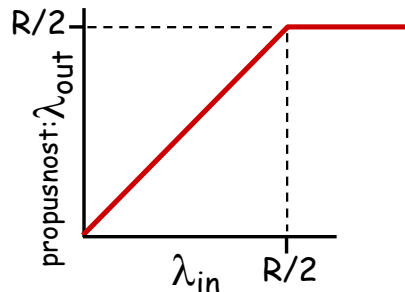
# Principi kontrole zagušenja

Najjednostavniji scenario:

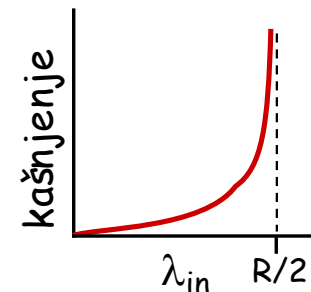
- router, beskonačni baferi
- Kapacitet linka  $R$
- Dva toka
- retransmisije nijesu potrebne



Šta se dešava kada se dolazna brzina  $\lambda_{in}$  približi  $R/2$ ?



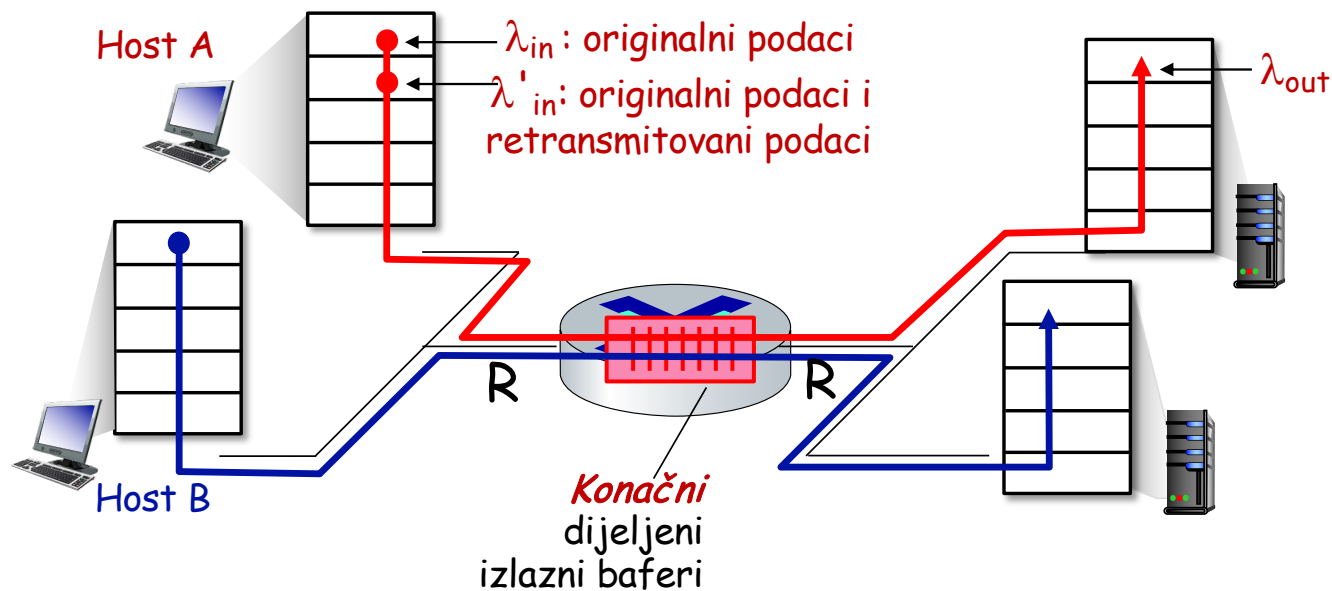
Maksimalna propusnost po konekciji:  $R/2$



Velika kašnjenja kada se  $\lambda_{in}$  približi kapacitetu

# Principi kontrole zagušenja

- jeda ruter konačnih bafera
- pošiljalac ponovo šalje izgubljene pakete ili pakete kojima je istekao time-out
  - ulaz nivoa aplikacija = izlaz nivoa aplikacije:  $\lambda_{in} = \lambda_{out}$
  - ulaz nivoa transporta uključuje retransmisije:  $\lambda'_{in} \geq \lambda_{in}$

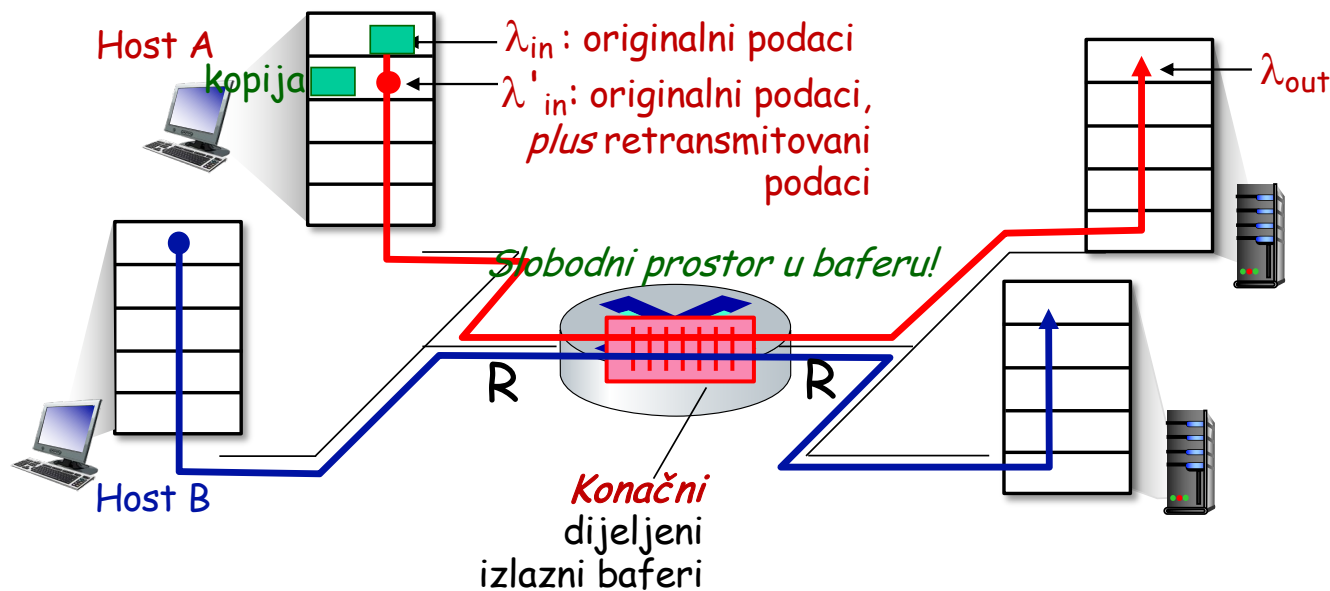
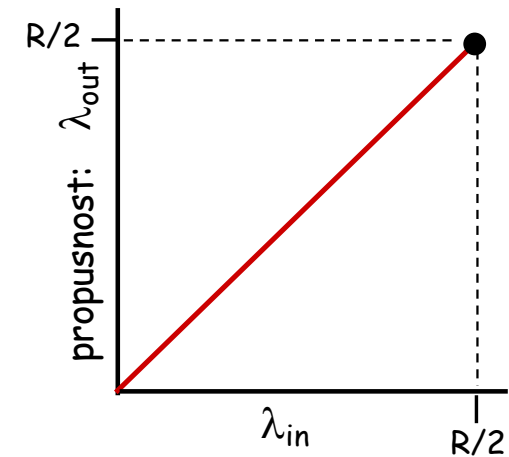




# Principi kontrole zagušenja

## Idealizacija: **perfektno znanje**

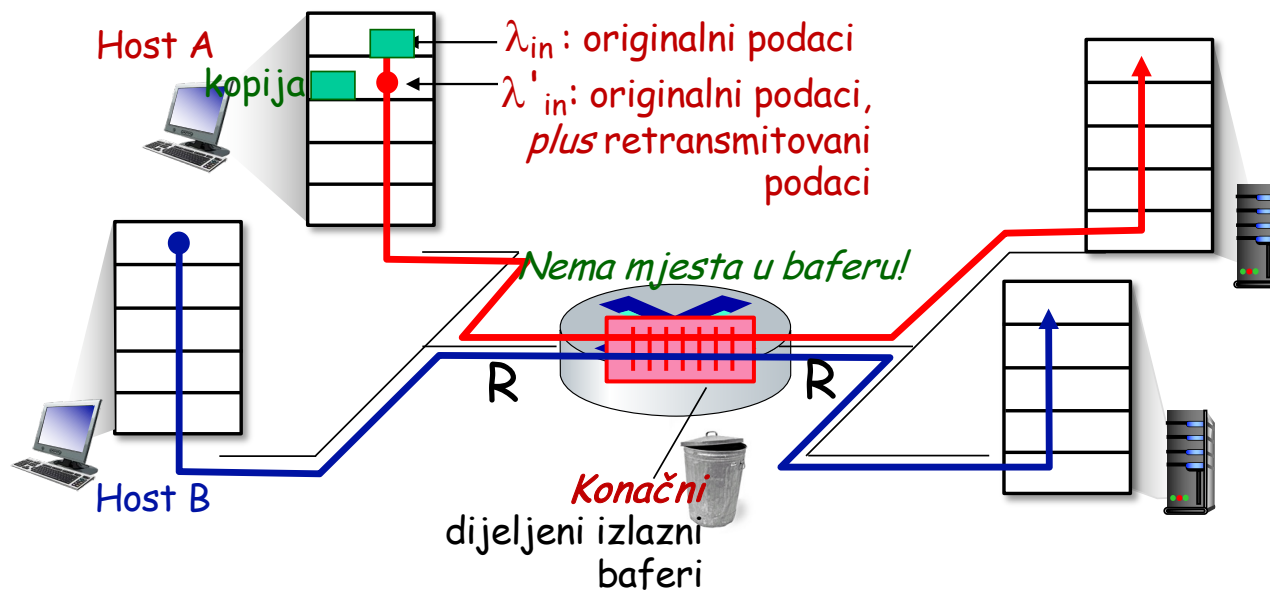
- Pošiljalac šalje samo onda kada u baferima ima mjesta



# Principi kontrole zagušenja

## Idealizacija: djelimično znanje

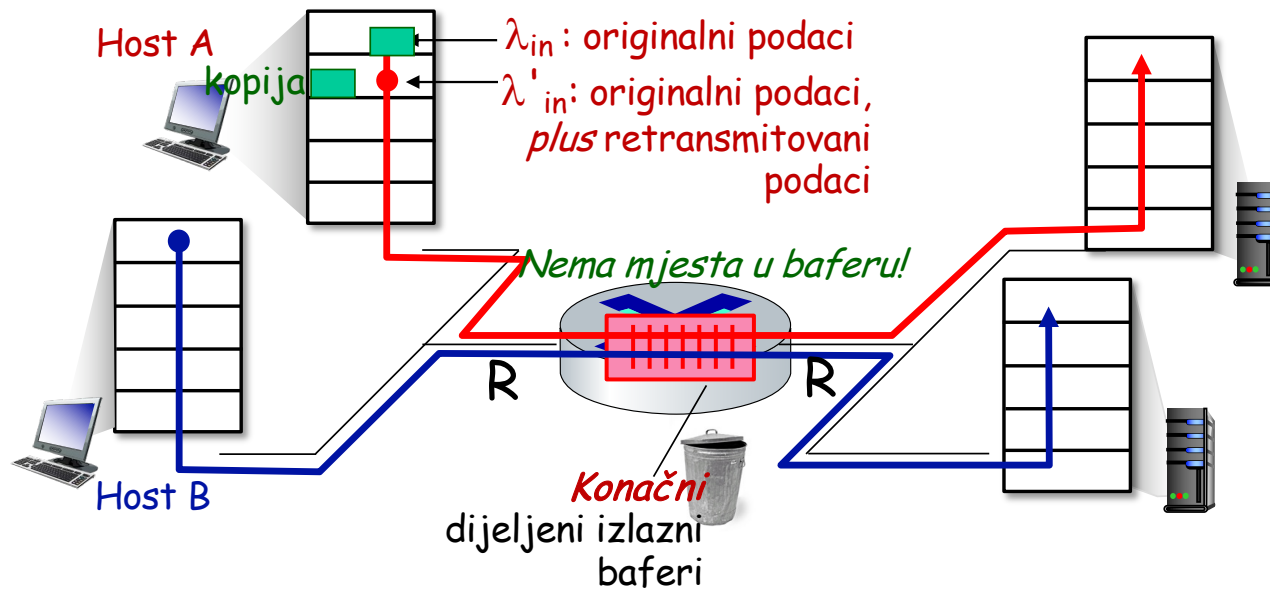
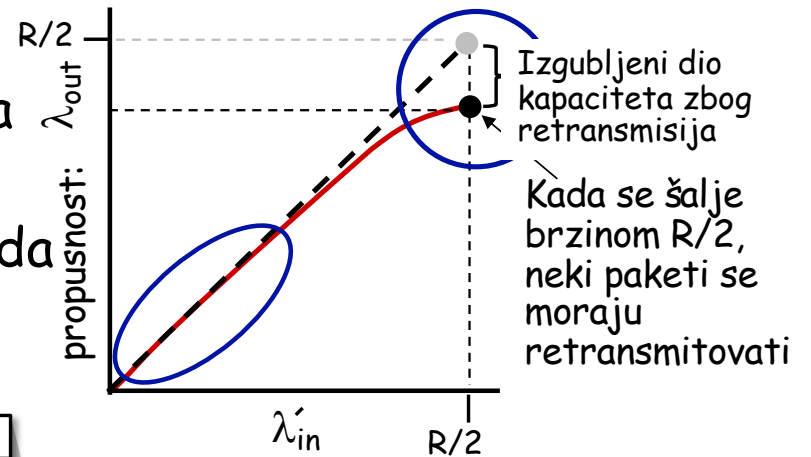
- ❑ Paketi mogu biti izgubljeni zbog nedostatka mjesta u baferu
- ❑ Pošiljalac zna kada je paket izgubljen tako da ponovo šalje samo izgubljene pakete



# Principi kontrole zagušenja

## Idealizacija: djelimično znanje

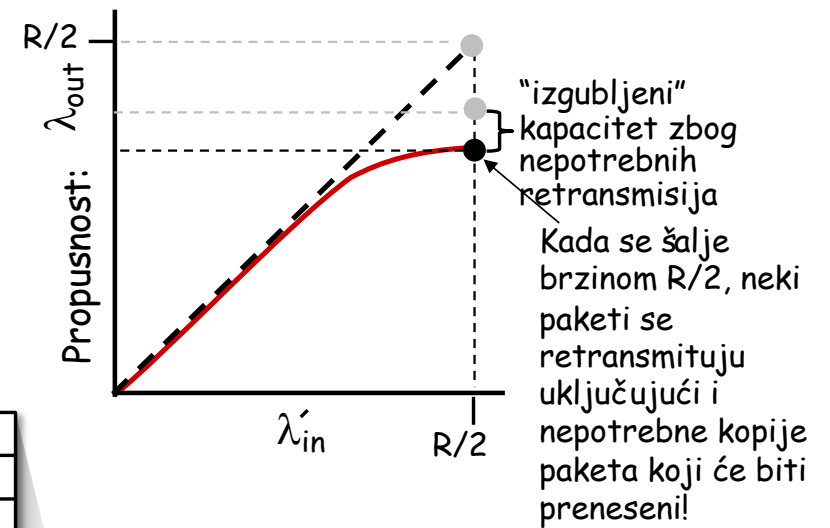
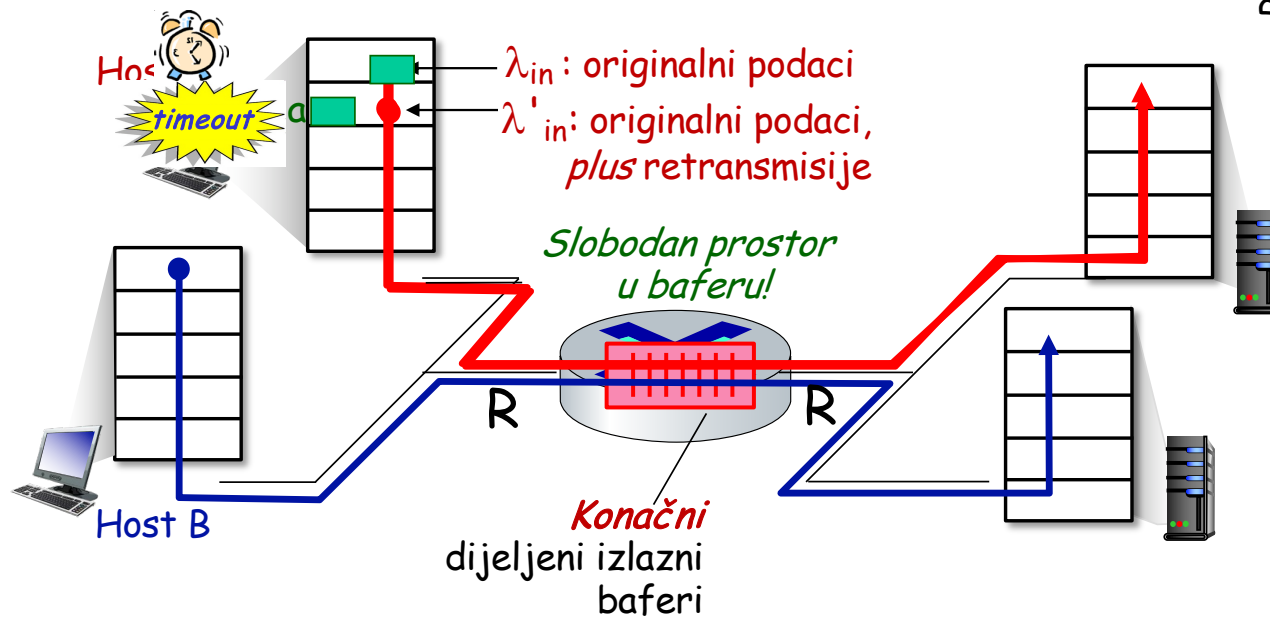
- ❑ Paketi mogu biti izgubljeni zbog nedostatka mjesta u baferu
- ❑ Pošiljalac zna kada je paket izgubljen tako da ponovo šalje samo izgubljene pakete



# Principi kontrole zagušenja

## Realni scenario: *nepotrebni duplikati*

- ❑ Paketi koji mogu biti izgubljeni zbog nedovoljno prostora u baferu zahtijevaju retransmisije
- ❑ Ali i pošiljačev timeout može isteći tako da izaziva retransmisiju paketa koji će stići



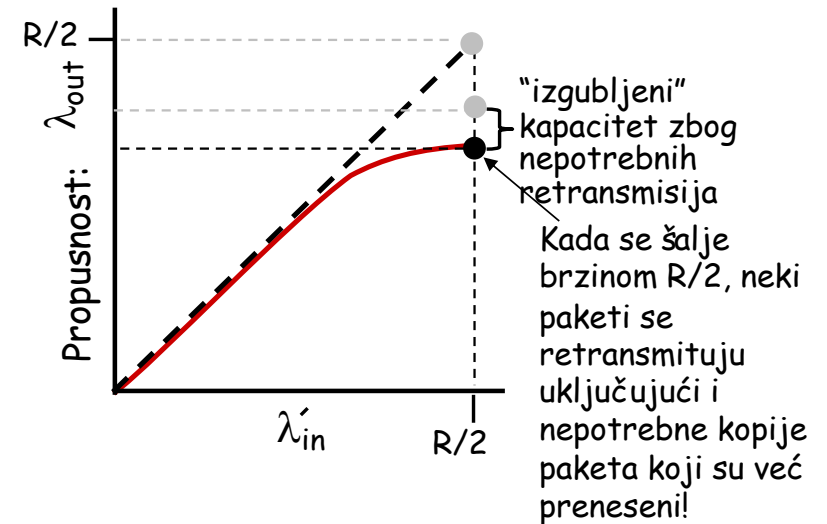
# Principi kontrole zagušenja

## Realni scenario: *nepotrebni duplikati*

- ❑ Paketi koji mogu biti izgubljeni zbog nedovoljno prostora u baferu zahtijevaju retransmisije
- ❑ Ali i pošiljačev timeout može isteći tako da izaziva retransmisiju paketa koji je već stigao

## Posljedice zagušenja:

- ❑ Potrebno je više retransmisija
- ❑ Javlja se i nepotrebne retransmisije koje smanjuju maksimalnu propusnost



# Principi kontrole zagušenja

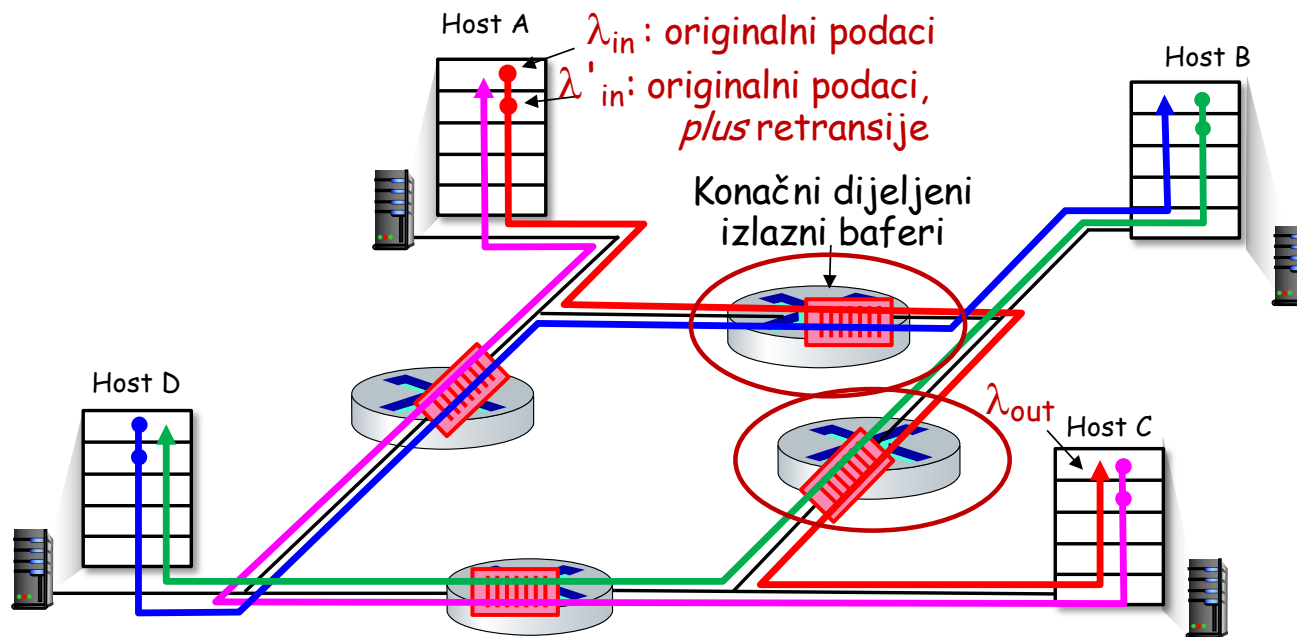
❑ 4 pošiljaoca

❑ više rutera

❑ timeout/retransmisija

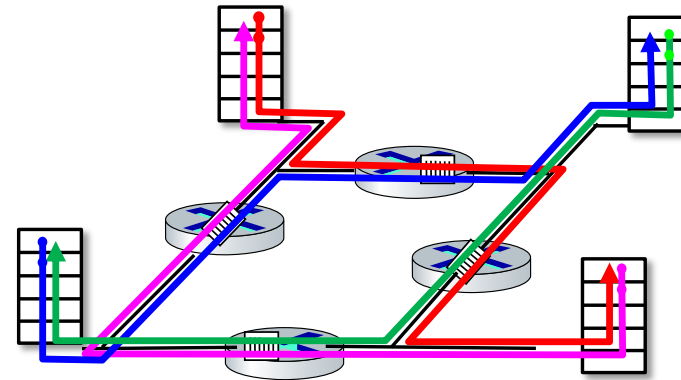
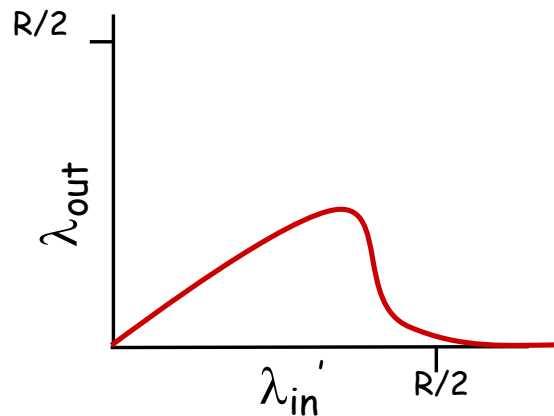
Šta se dešava kada  $\lambda_{in}$  i  $\lambda_{in}'$  porastu?

Kada crveni  $\lambda_{in}'$  raste, svi plavi paketi na gornjem redu čekanja se odbacuju smanjujući plavu propusnost na 0.



# Principi kontrole zagušenja

---



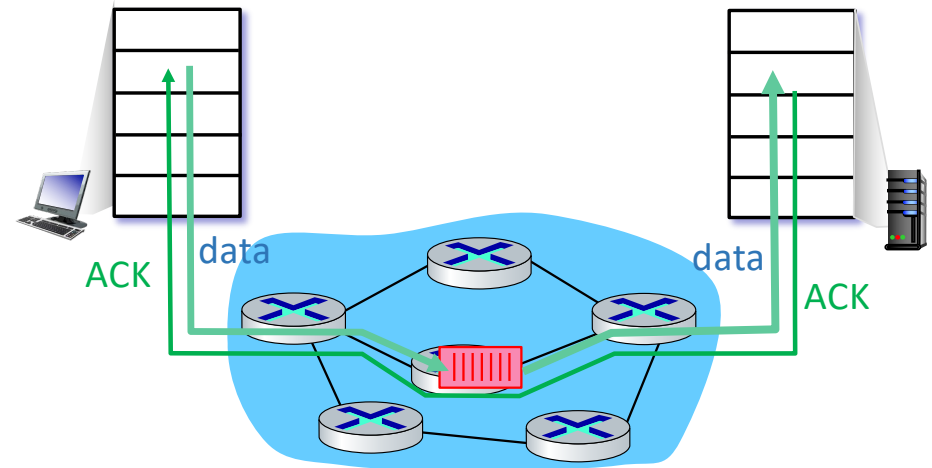
Još jedna posledica kolizije:

- ❑ Kada se paket odbaci, svi resursi do tada iskorišćeni za prenos paketa su izgubljeni!

# Principi kontrole zagušenja

## Kontrola od kraja do kraja:

- ❑ Nema eksplicitne informacije o zagušenju od mreže
- ❑ Spoznaja o zagušenju se dobija praćenjem gubitaka, kašnjenja, ...
  - TCP

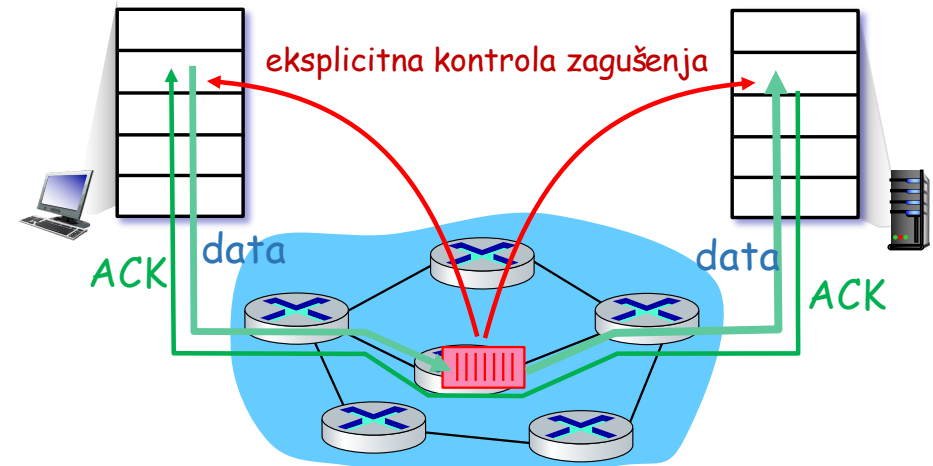




# Principi kontrole zagušenja

## Kontrola zagušenja uz asistenciju mreže:

- ❑ ruter obezbjeđuje hostovima, čiji saobraćaj se prenose preko njega, eksplicitnu informaciju o zagušenju
- ❑ može poslati informaciju o nivou zagušenja ili brzinu prenosa kojom host može da šalje podatke
  - ❑ TCP ECN, ATM, DECbit,...



# Glava 3: Sadržaj

---

- 3.1 Servisi nivoa transporta
- 3.2 Multipleksiranje i demultipleksiranje
- 3.3 Nekonektivni transport: UDP

## 3.4 Konektivni transport: TCP

- Struktura segmenta
- Pouzdani prenos podataka
- Kontrola protoka
- Upravljanje vezom

## 3.5 Principi kontrole zagušenja

## 3.6 TCP kontrola zagušenja

## 3.7 QUIC

# TCP kontrola zagušenja

---

- Kontrola od kraja do kraja (bez učesća mreže)
- Pošiljalac ograničava slanje:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

- Približno,

$$\text{brzina} = \frac{\text{cwnd}}{\text{RTT}} \quad \text{b/s}$$

- `cwnd` je dinamička funkcija detekcije zagušenja mreže

# TCP kontrola zagušenja: AIMD

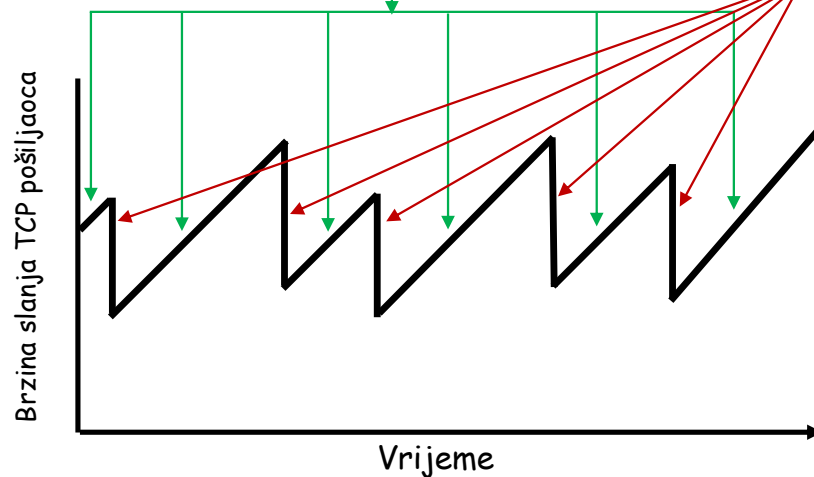
*Pristup:* pošiljaoci povećavaju brzinu prenosa dok se ne desi gubitak paketa (zagušenje), kada se smanjuje brzina na polovinu brzine u trenutku gubitka paketa

## Aditive Increase

Povećava brzinu za 1 MSS nakon svakog RTT-a u kome se neijesu pojavili gubici

## Multiplicative Decrease

Smanjuje brzinu na pola nakon RTT u kome se pojavio gubitak



**AIMD** "testera":  
*provjera* dostupnog kapaciteta

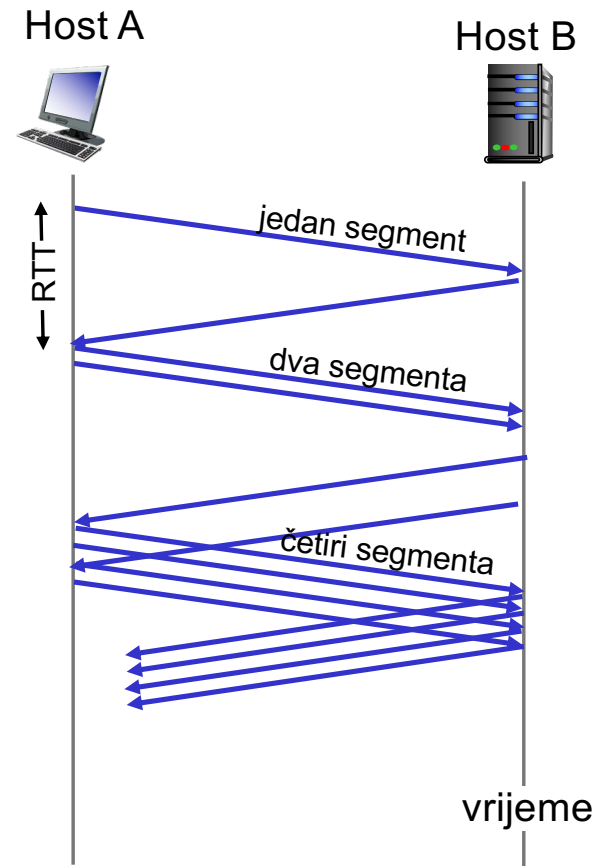
# TCP kontrola zagušenja: Slow start

---

- Kada se konekcija uspostavi, CongWin = 1 MSS
  - Primjer: MSS = 500 B & RTT = 200 ms
  - Inicijalna brzina = 20 kb/s
- Dostupna propusnost može biti mnogo veća od MSS/RTT
  - Poželjno je brzo podešavaje na željenu brzinu
- Kada se konekcija uspostavi, brzina se povećava eksponencijalno do prvog gubitka

# TCP kontrola zagušenja: Slow start

- Kada se konekcija uspostavi, brzina se eksponencijalno povećanje do prvog gubitka :
  - Inicijalna vrijednost  $cwnd=1$
  - Udvostručuje se  $cwnd$  svakog RTT-a tako što  $cwnd$  inkrementira sa svakim primljenim ACK
  - Inicijalna brzina je niska ali brzo raste



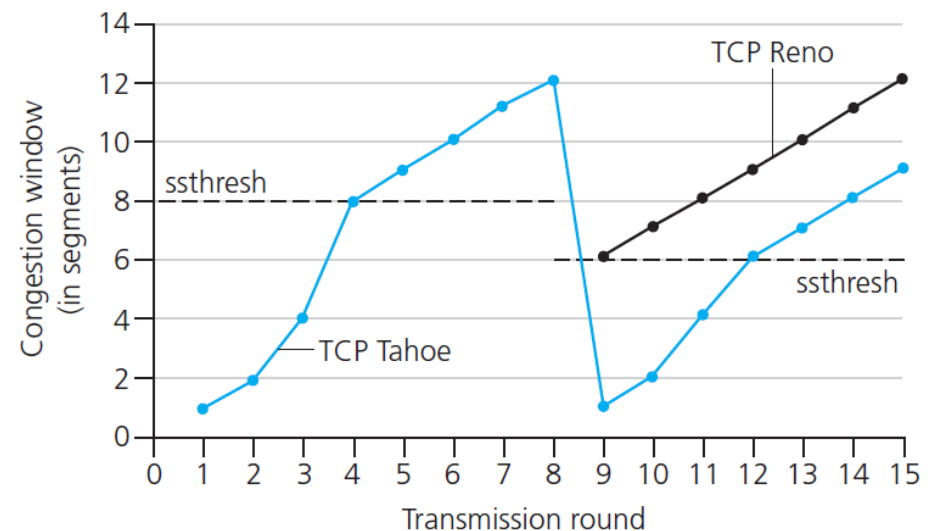
# TCP kontrola zagušenja

Kada eksponencijalni rast prelazi u linearni?

Eksponencijalni rast prelazi u linearni kad cwnd dostigne 1/2 svoje vrijednosti u trenutku detekcije zagušenja

## Implementacija:

- ❑ Varijabilni ssthresh (Tahoe)
- ❑ U slučaju gubitka, ssthresh se postavlja na 1/2 vrijednosti CongWin prije gubitka
- ❑ U slučaju gubitka cwnd se smanjuje na pola (Reno) a cwnd eksponencijalno raste



# TCP kontrola zagušenja: Tahoe

---

- ❑ "Slow Start", izbjegavanje kolizije
- ❑ Detektuje zagušenje kroz isticanje timeout-a i trostruki prijem isovjetne potvrde
- ❑ Inicijalizacija
  - $cwnd = 1$ ;
  - $ssthresh = 1/2 \text{ Max}(cwnd)$
- ❑ Poslije timeouta i prijema trostruke istovjetne potvrde
  - $ssthresh = 1/2 cwnd$ ,  $cwnd = 1$
  - Ulazi u slow start



# TCP kontrola zagušenja:Reno

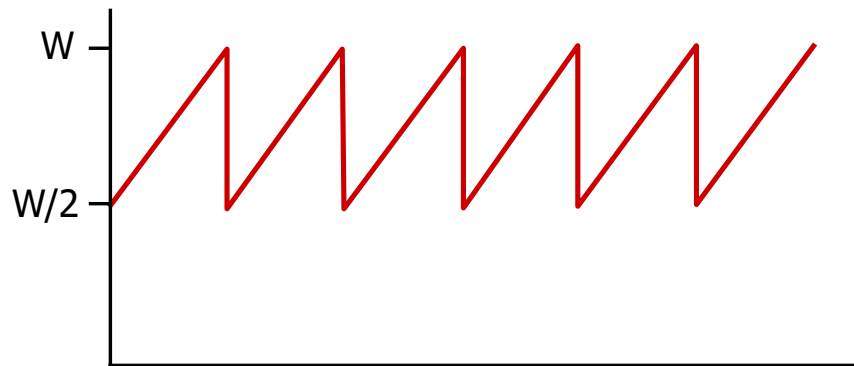
---

- ❑ "Fast Retransmit", "Fast recovery"
- ❑ Detektuje zagušenje kroz timeout-e i prijem trostrukih istovjetnih potvrda
- ❑ Kada se tri puta primi ista potvrda
  - Izbjegava slow start i ide direktno u fazu izbjegavanja kolizije
  - $ssthresh = 1/2 cwnd$ ;  $cwnd = ssthresh$
  - (Koristi AIMD)
- ❑ Kada se pojavi timeout
  - "Slow start"

# TCP kontrola zagušenja: Reno

Koliko iznosi srednja propusnost TCP-a u funkciji veličine prozora i RTT?

- Ignoriše se slow start
- Neka je  $W$  veličina prozora kada nastaju gubici.
- Kada je veličina prozora  $W$ , propusnost je  $W/RTT$
- Poslije gubitka, veličina prozora pada na  $W/2$ , propusnost na  $W/2RTT$ .
- Srednja propusnost:  $.75 W/RTT$

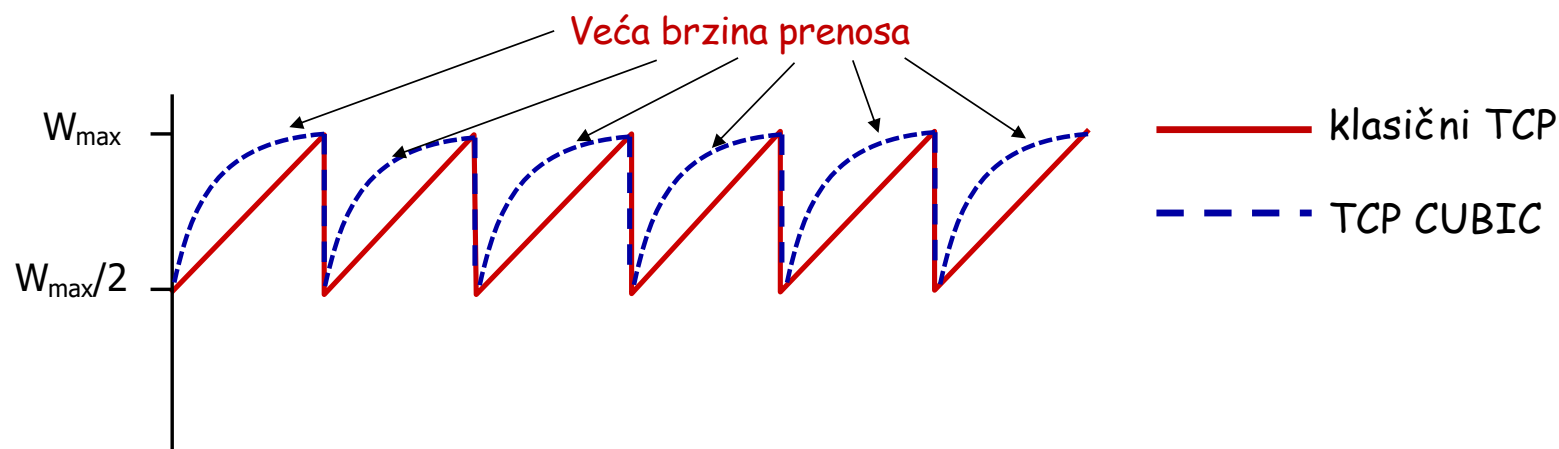


# TCP kontrola zagušenja: CUBIC

Ima li boljeg načina od AIMD za provjeru dostupnog kapaciteta?

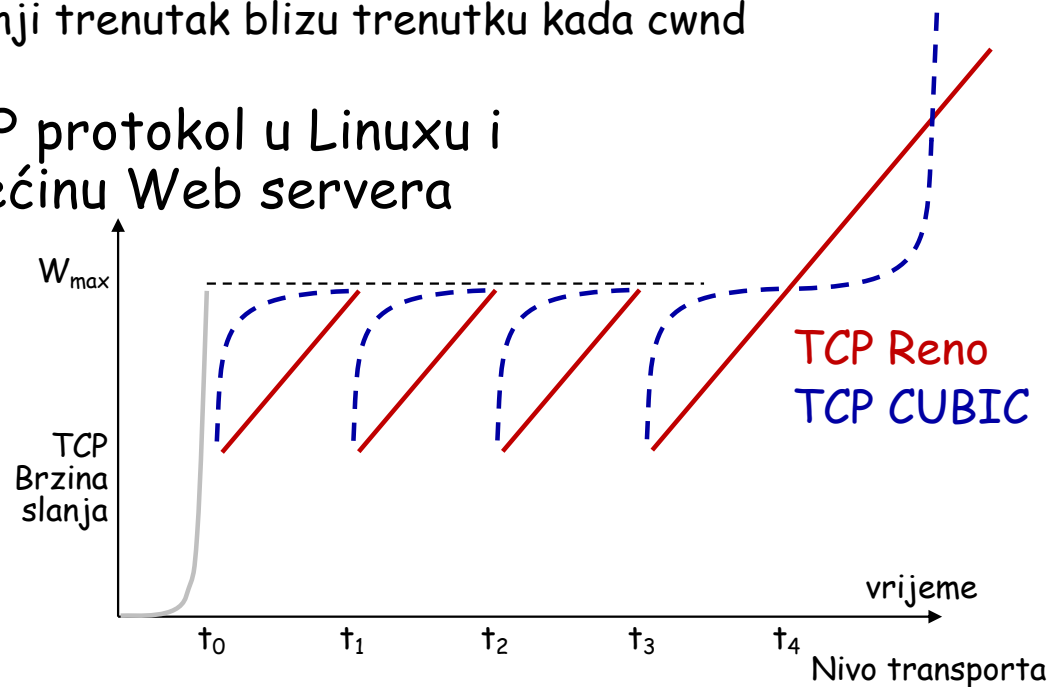
## □ Intuicija:

- $W_{\max}$ : brzina slanja pri kojoj se detektuje kolizija
- stepen zagušenosti bottleneck linka se vjerovatno ne mijenja mnogo
- poslije smanjenja brzine na pola pri pojavi gubitka treba brže povećavati cwnd a pri približavanju  $W_{\max}$  sporije



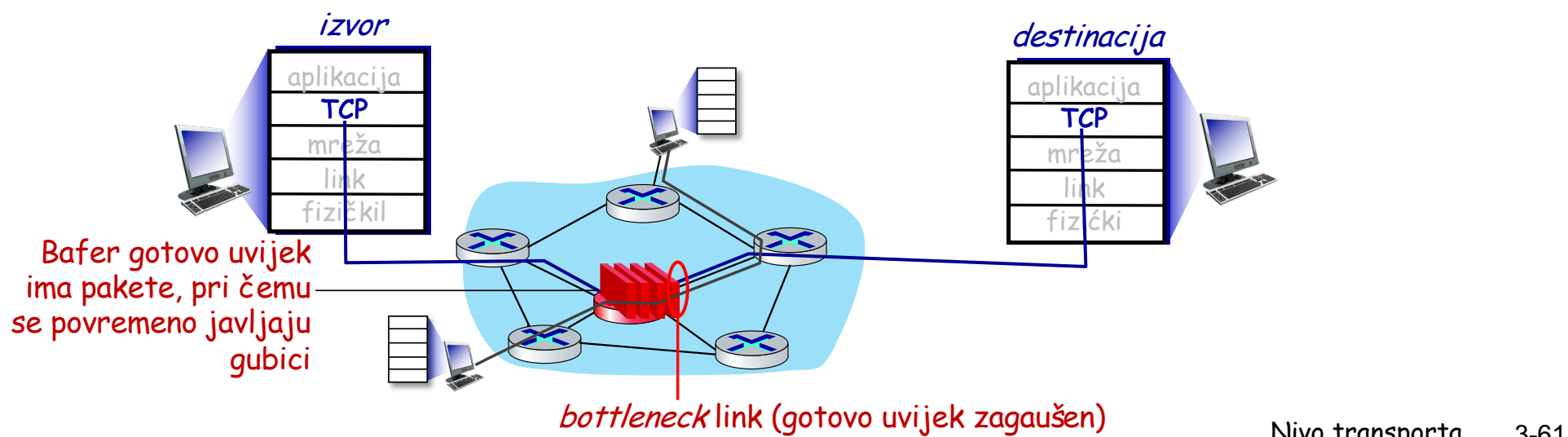
# TCP kontrola zagušenja: CUBIC

- Vrijeme kada TCP cwnd dostiže  $W_{max}$  se može podešavati povećanjem cwnd kao kubne funkcije razlike između sadašnjeg trenutka i trenutka kada cwnd dostiže  $W_{max}$ 
  - veće povećanje ako je sadašnji trenutak daleko od trenutka kada cwnd dostiže  $W_{max}$
  - manje povećanje ako je sadašnji trenutak blizu trenutku kada cwnd dostiže  $W_{max}$
- TCP CUBIC je default TCP protokol u Linuxu i najpopulararniji TCP za većinu Web servera



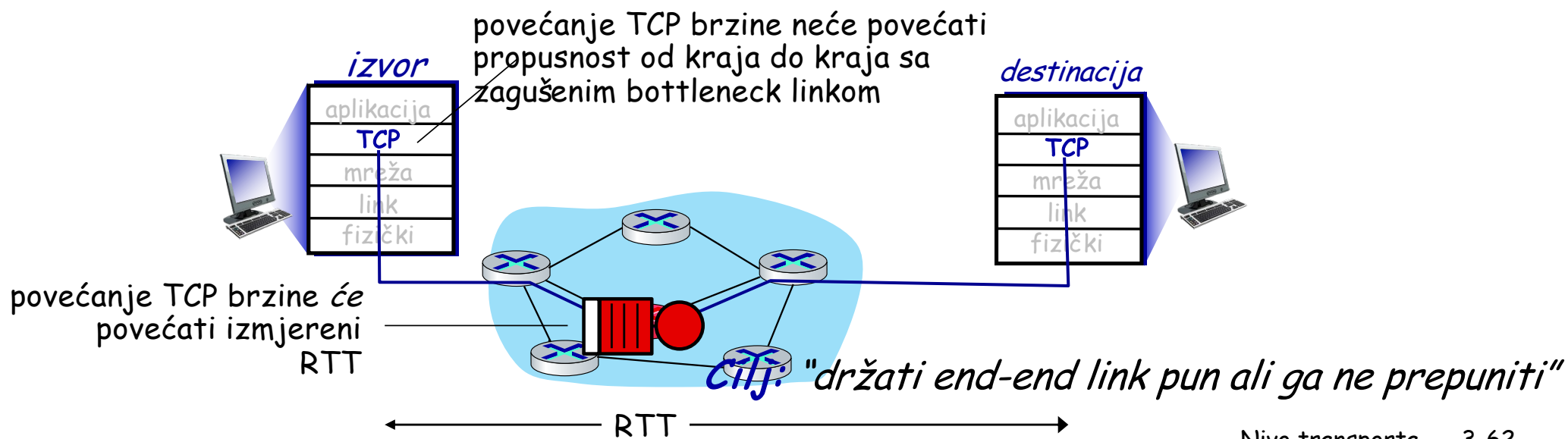
# TCP kontrola zagušenja: CUBIC

TCP (clasični i CUBIC) povećavaju brzinu slanja dok se ne pojavi gubitak paketa na izlazu nekog rutera (*bottleneck link*)



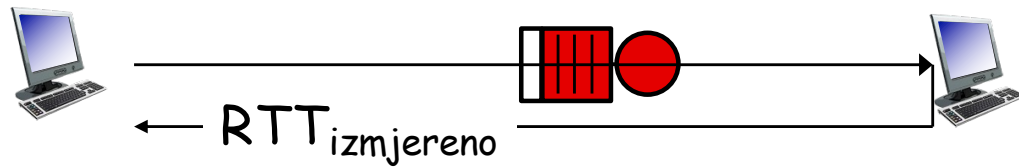
# TCP kontrola zagušenja: CUBIC

- ❑ TCP (clasični, CUBIC) povećavaju brzinu slanja dok se ne pojavi gubitak paketa na izlazu nekog rutera (bottleneck link)
- ❑ Radi razumijevanja zagušenja bitno je fokusirati se na bottleneck link



# TCP kontrola zagušenja bazirana na kašnjenju

Održavanjem TCP toka "dovoljno punim ali ne prepunim" *bottleneck* link se održava iskorišćenim uz izbjegavanje velikih kašnjenja



$$\text{Izmjerena propusnost} = \frac{\# \text{ B-ta poslatih u poslednjem RTT}}{\text{RTT}_{\text{izmjereno}}}$$

## Pristup baziran na kašnjenju:

- $\text{RTT}_{\min}$  - minimalni izmjereni RTT na nezagušenoj ruti
- Propusnost na nezagušenoj ruti sa prozorom zagušenja  $\text{cwnd}$  je  $\text{cwnd}/\text{RTT}_{\min}$

If (izmjerena propusnost je "vrlo bliska" propusnosti kada nema zagušenja)  
    linearno smanjiti  $\text{cwnd}$                       /\* jer ruta nije zagušena\*/

else if (izmjerena propusnost je "daleko ispod" propusnosti kada nema zagušenja)  
    linearno povećati  $\text{cwnd}$                     /\* jer je ruta zagušena \*/

# TCP kontrola zagušenja bazirana na kašnjenju

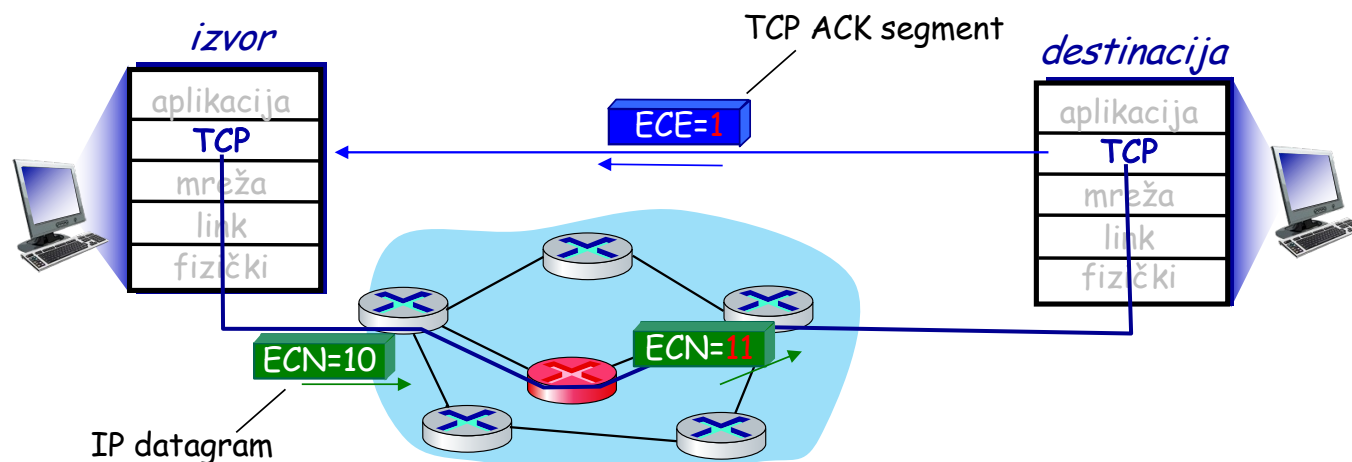
- ❑ Kontrola zagušenja bez forsiranja pojave zagušenja
- ❑ Maksimizacija propusnosti održavanjem TCP toka "dovoljno punim" uz držanje kašnjenja na niskom nivou "... ali ne prepunim"
- ❑ Veliki broj primijenjenih verzija TCP-a koriste ovaj pristup
  - ❑ *Bottleneck Bandwidth and Round-trip propagation time (BBR)* se koristi na Google-ovoj (internoj) okosnici mreže



# EksPLICITNA notifikacija zagušenja

Neke varijante TCP-a su često bazirane na kontroli zagušenja uz asistenciju mreže:

- ❑ *Explicit Congestion Notification* (ECN)
- ❑ Ruter koristi dva bita ToS polja u zaglavlju IP paketa (ECN) da markira zagušenje pri čemu mrežni operator određuje politiku markiranja
- ❑ Indikacija zagušenja se nosi do destinacije
- ❑ Destinacija postavlja *ECN-Echo* (ECE) bit u ACK segmentu kako bi obavijestila izvor o zagušenju
- ❑ Uključuje IP (ECN bite u IP zaglavlju) i TCP (ECE bit u TCP zaglavlju)



# TCP kod dugačkih i brzih ruta

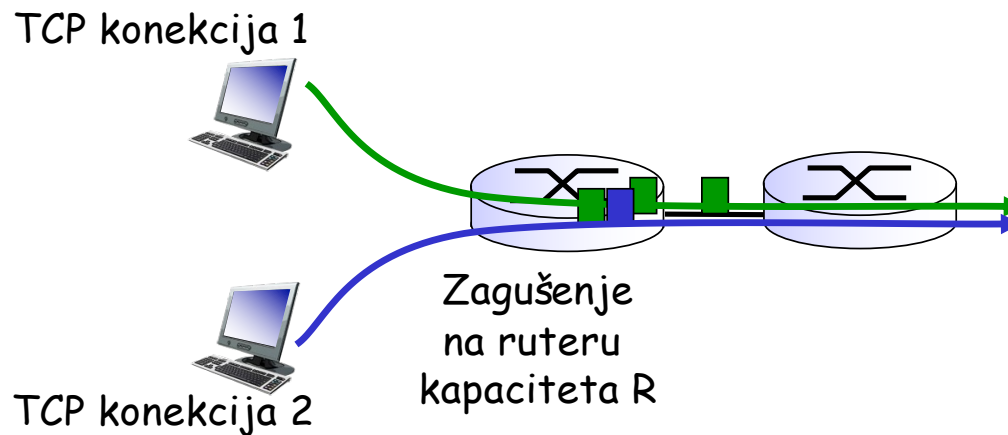
- Primjer: 1500 B segmenti, 100ms RTT, želi se 10 Gb/s propusnost
- Zahtijeva se veličina prozora od  $W = 83,333$  segmenata
- Srednja propusnost u funkciji vjerovatnoće gubitka:

$$\frac{1,22 MSS}{RTT \sqrt{L}}$$

- Vjerovatnoća gubitka  $\rightarrow L = 2 \cdot 10^{-10}$
- Potrebne su verzije TCP-a za *high-speed* potrebe!

# Korektnost TCP-a

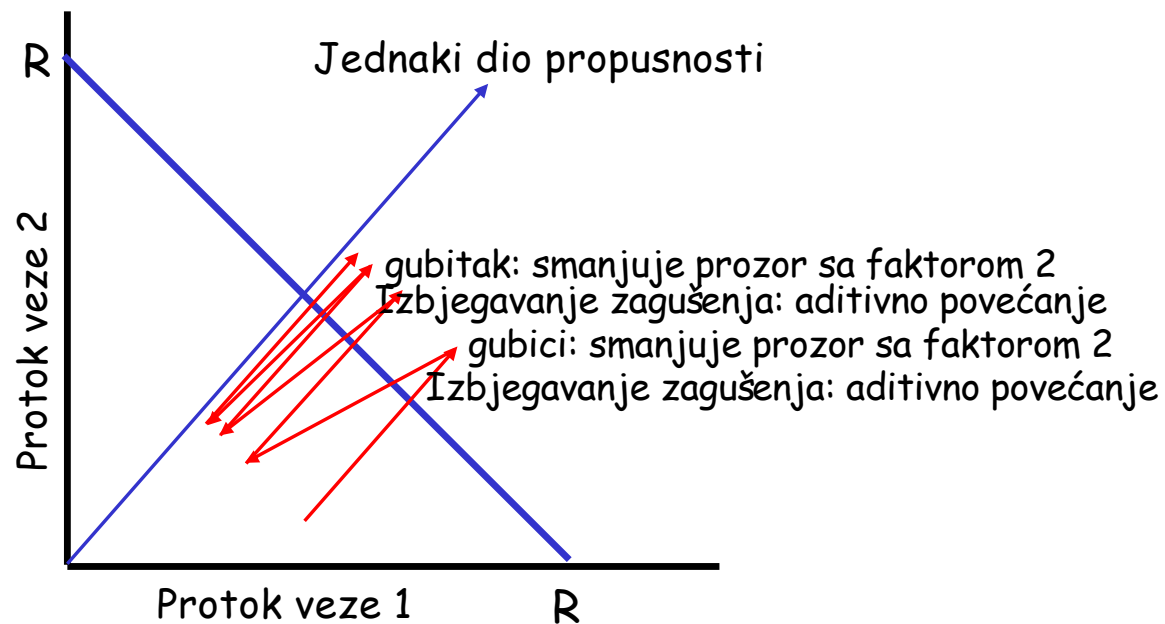
Ako  $K$  paralelnih TCP sesija dijele isti zagušeni link brzine prenosa  $R$ , svaki bi trebao da ima srednju propusnost od  $R/K$



# Korektnost TCP-a

Dvije sučeljene sesije:

- Aditivno povećanje daje porast za 1, tako da protok raste
- Multiplikativno smanjenje smanjuje protok proporcionalno



*Da li je TCP fer?*

Da, u idealizovanim uslovima:

- Isti RTT
- Fiksni broj sesija u fazi izbjegavanja kolizije

# Korektnost TCP-a

---

## Korektnost i UDP

- ❑ Multimedijalne aplikacije često ne koriste TCP
  - Ne želi se da kontrola zagušenja ograniči kapacitet
- ❑ Umjesto toga se koristi UDP:
  - Ubacuje audio/video konstantnom brzinom, toleriše gubitak paketa
- ❑ Nema neke „Internet policije“ koja bi kontrolisala korišćenje kontrole zagušenja

## Korektnost i paralelne TCP konekcije

- ❑ Nema prevencije da aplikacija otvori paralelne veze između 2 hosta.
- ❑ *Web browser*-i to rade
- ❑ Primjer: link brzine prenosa  $R$  podržava 9 konekcija;
  - Nova aplikacija pita za jednu TCP konekciju i dobija propusnost od  $R/10$
  - Nova aplikacija pita za 11 novih TCP konekcija, i dobija više od  $R/2$ !

# Budućnost TCP-a

- TCP, UDP: ključni protokoli prethodnih 40 godina
- Različite verzije TCP protokola su razvijene za specifične scenarije:

Scenario	Izazovi
Dugačke i brze rute (prenos velikih podataka)	Mnogo paketa se prenosi tako da gubici mogu oboriti link
Bežične mreže	Gubici zbog bežičnih linkova i mobilnosti što TCP tretira kao gubitke zbog zagušenja.
Linkovi sa velikim kašnjenjem	Ekstremno dugački RTT
Mreže data centara	Osjetljivost na kašnjenje
<i>Background</i> saobraćajni tokovi	<i>Background</i> TCP tokovi niskog prioriteta

- Seljenje transportnih funkcija na nivo aplikacije uz korišćenje UDP
  - HTTP/3: QUIC

# Glava 3: Sadržaj

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

3.4 Konektivni transport: TCP

- Struktura segmenta
- Pouzdani prenos podataka
- Kontrola protoka
- Upravljanje vezom

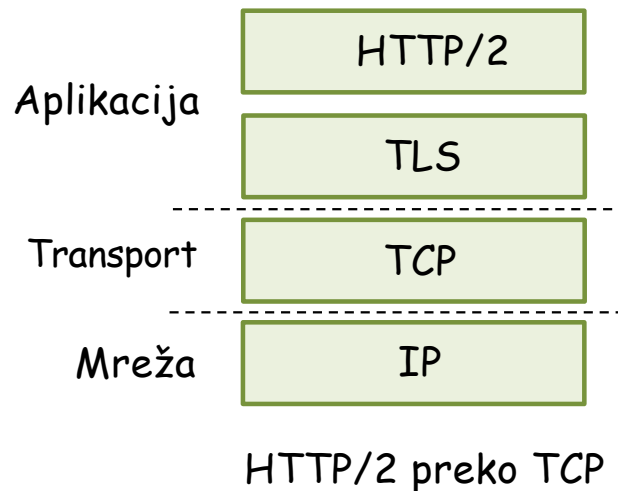
3.5 Principi kontrole zagušenja

3.6 TCP kontrola zagušenja

3.7 QUIC

# QUIC: Quick UDP Internet Connections

- Protokol nivoa aplikacije koji koristi UDP
  - Poboljšava performanse HTTP protokola
  - Implementiran na mnogim Google-ovim serverima, aplikacijama (Chrome, mobilna YouTube aplikacija,...)



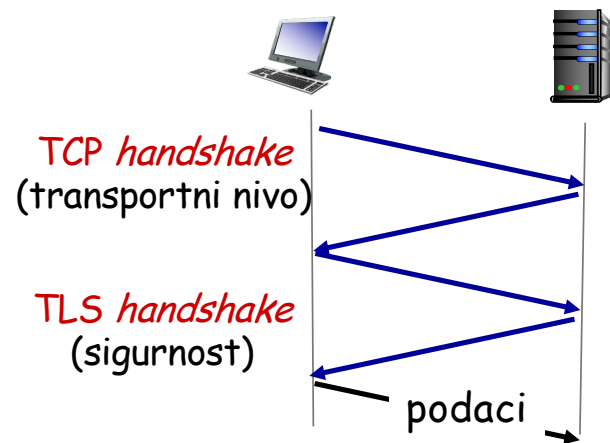


# QUIC: Quick UDP Internet Connections

Obezbjeđuje uspostavljanje konekcije, kontrolu greške, kontrolu zagušenja,...

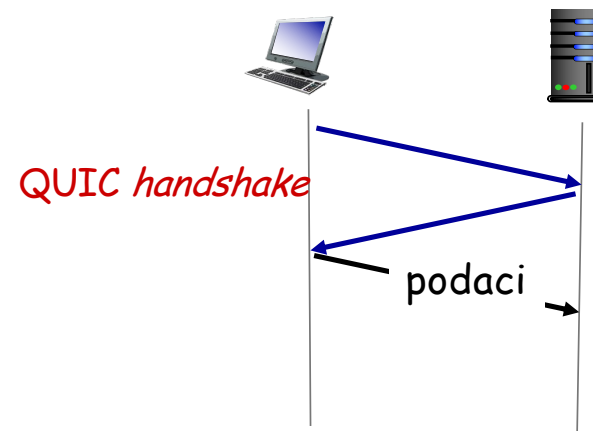
- ❑ Algoritmi kontrole greške i zagušenja su slični TCP-evim algoritmima
- ❑ Konektivnost: pouzdanost, kontrola zagušenja, autentikacija, enkripcija, uspostavljanje u jednom RTT-u
- ❑ Više strimova nivoa aplikacije se multipleksiraju preko jedne QUIC konekcije uz
  - Odvojene zaštite i mehanizme pouzdanog prenosa
  - Zajedničku kontrolu zagušenja

# QUIC: Uspostavljanje konekcije



TCP (pouzdanost, kontrola zagušenja) + TLS (autentikacija, sigurnost)

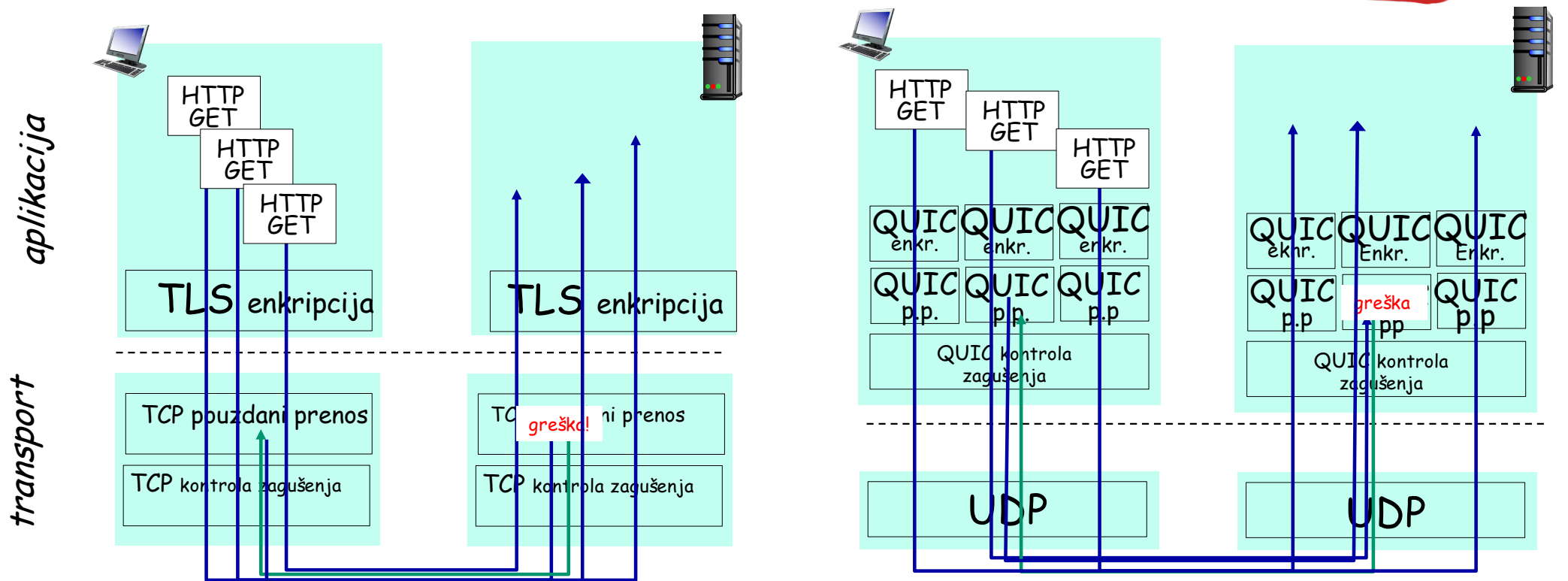
□ Dva handshake-a



QUIC: pouzdanost, kontrola zagušenja, autentikacija, sigurnost

□ Jedan handshake

# QUIC: paralelno slanje streamova bez HOL blokiranja



(a) HTTP 1.1

(b) HTTP/3 sa QUIC: bez HOL blokiranja