

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 i kontrola zagušenja

3. Nivo transporta

3.1 *Servisi nivoa transporta*

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

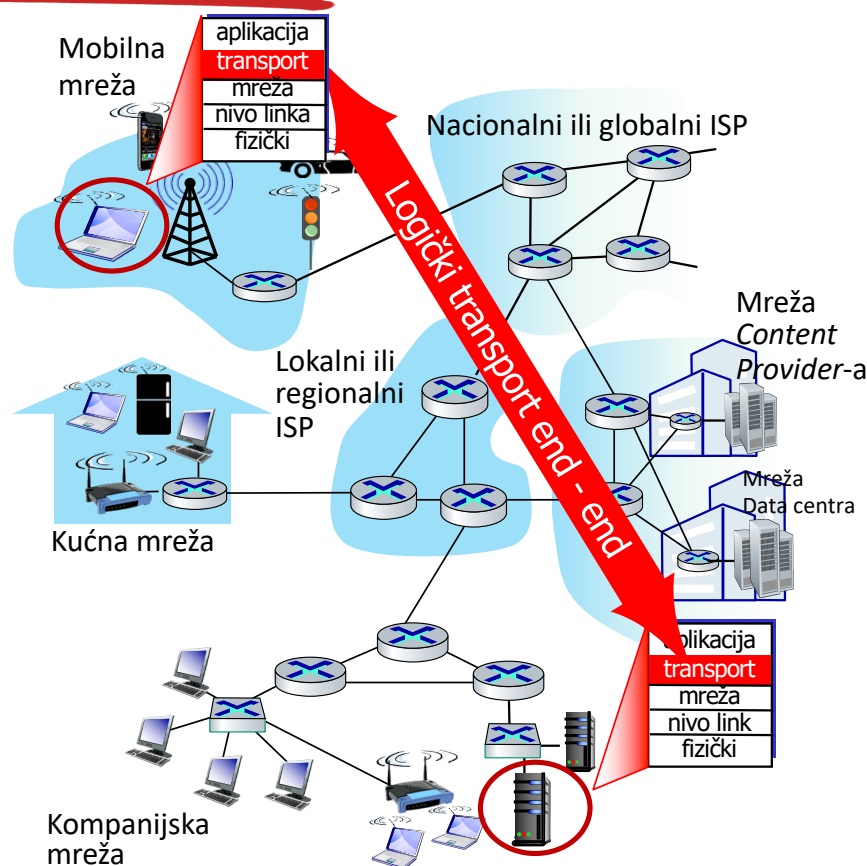
3.4 Principi pouzdanog prenosa podataka

3.5 Konektivni transport: TCP

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

Transportni servisi i protokoli

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



Transportni servisi i protokoli

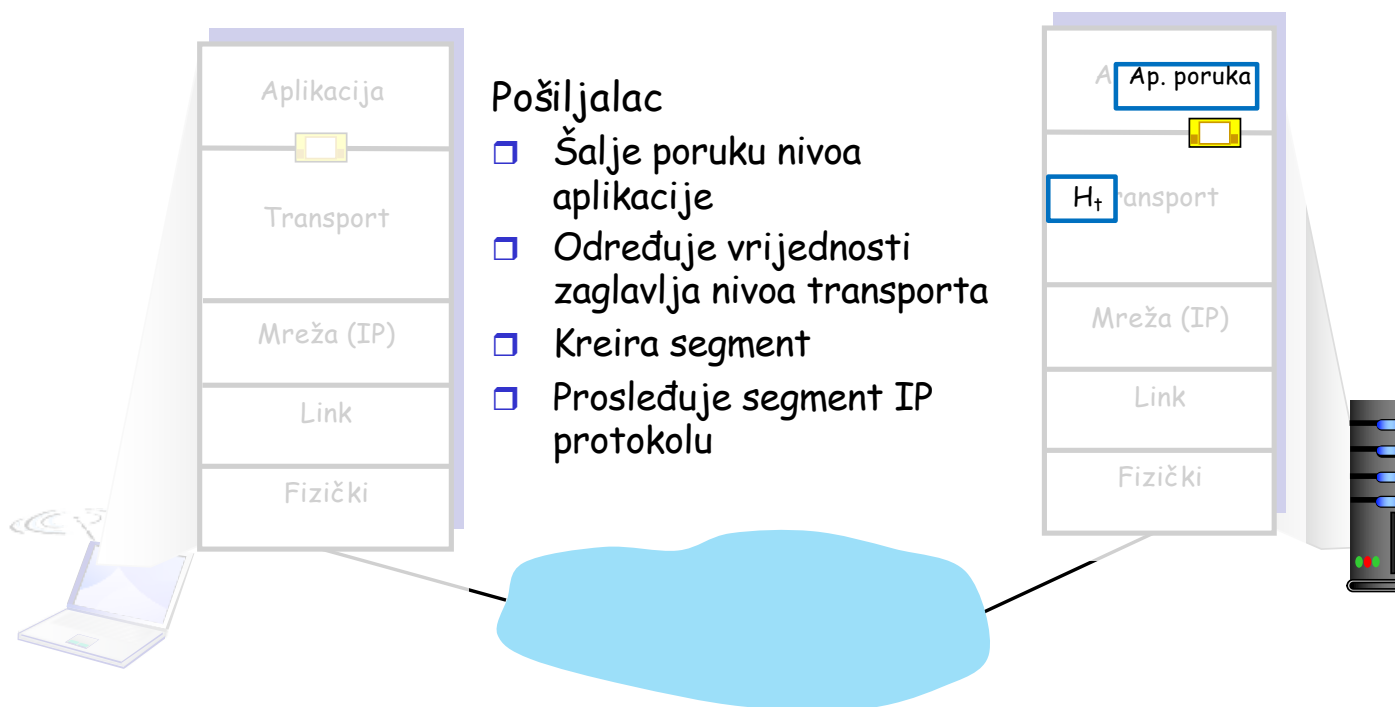
- *Mrežni nivo:* logička komunikacija između hostova
- *Transportni nivo:* **logička komunikacija između procesa**
 - Oslanja se na servise mrežnog nivoa i poboljšava njihove osobine

Analogija:

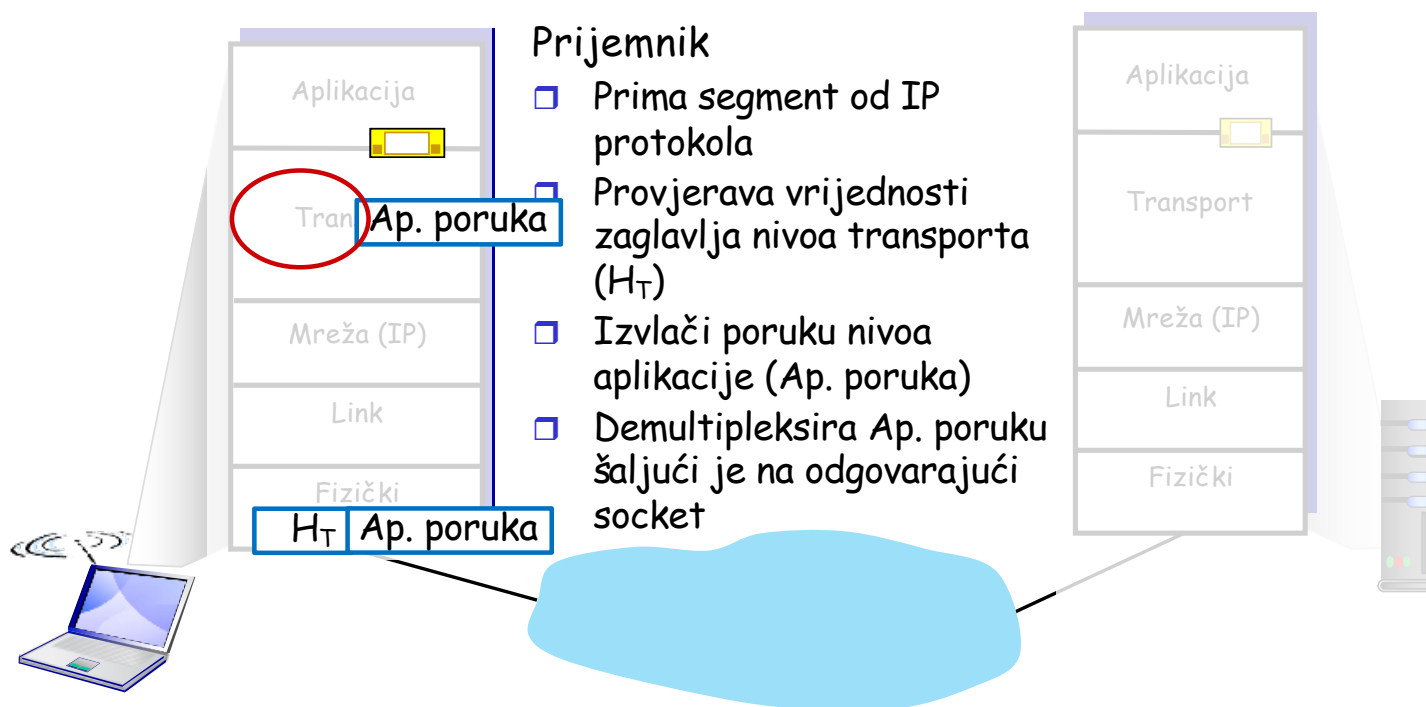
12 ljudi šalje pisma za 12 ljudi

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

Transportni servisi i protokoli

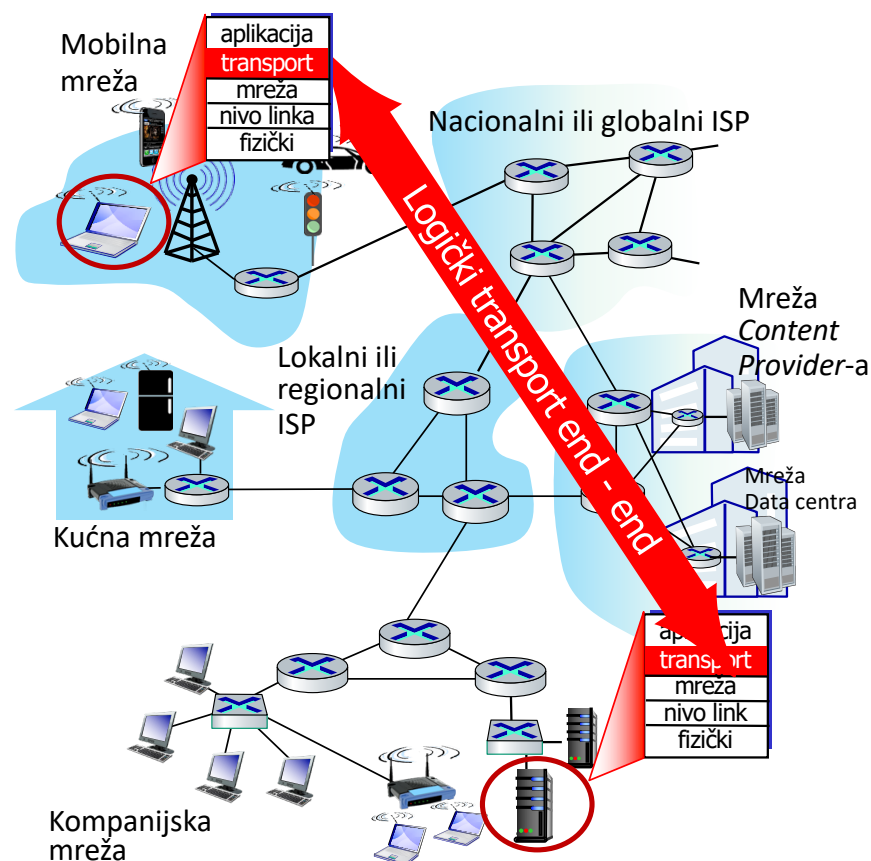


Transportni servisi i protokoli



Transportni servisi i protokoli

- ❑ TCP (*Transmission Control Protocol*)
 - Pouzdana, redosledna isporuka bajta
 - Kontrola zagušenja
 - Kontrola protoka
 - Uspostavljanje konekcije
- ❑ UDP (*User Datagram Protocol*)
 - Nepouzdana, neredosledna isporuka segmenata
 - Bez unapređenja “best-effort” IP servisa
- ❑ Servisi koji se ne pružaju:
 - Garantovano kašnjenje
 - Garantovana propusnost



3. Nivo transporta

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

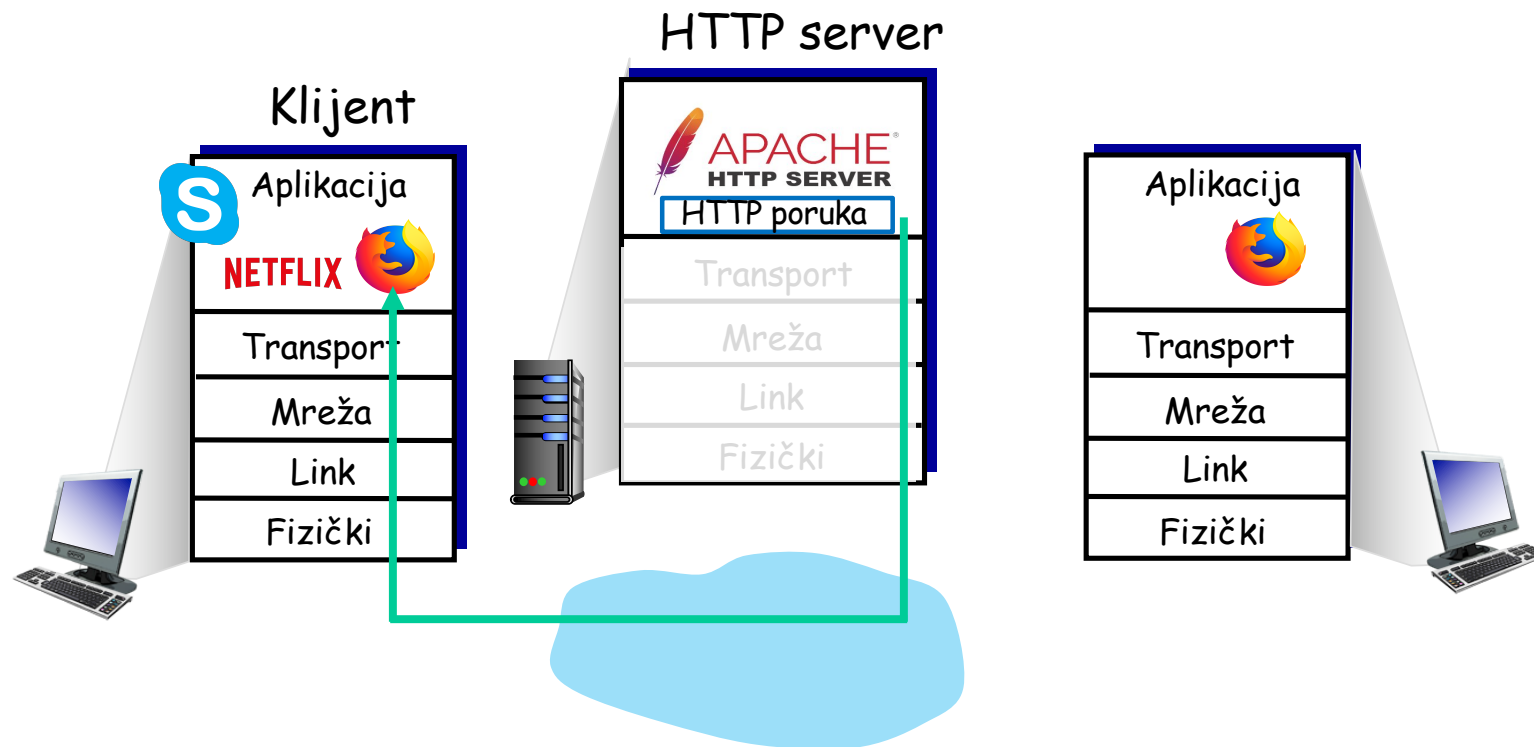
3.3 Nekonektivni transport: UDP

3.4 Principi pouzdanog prenosa podataka

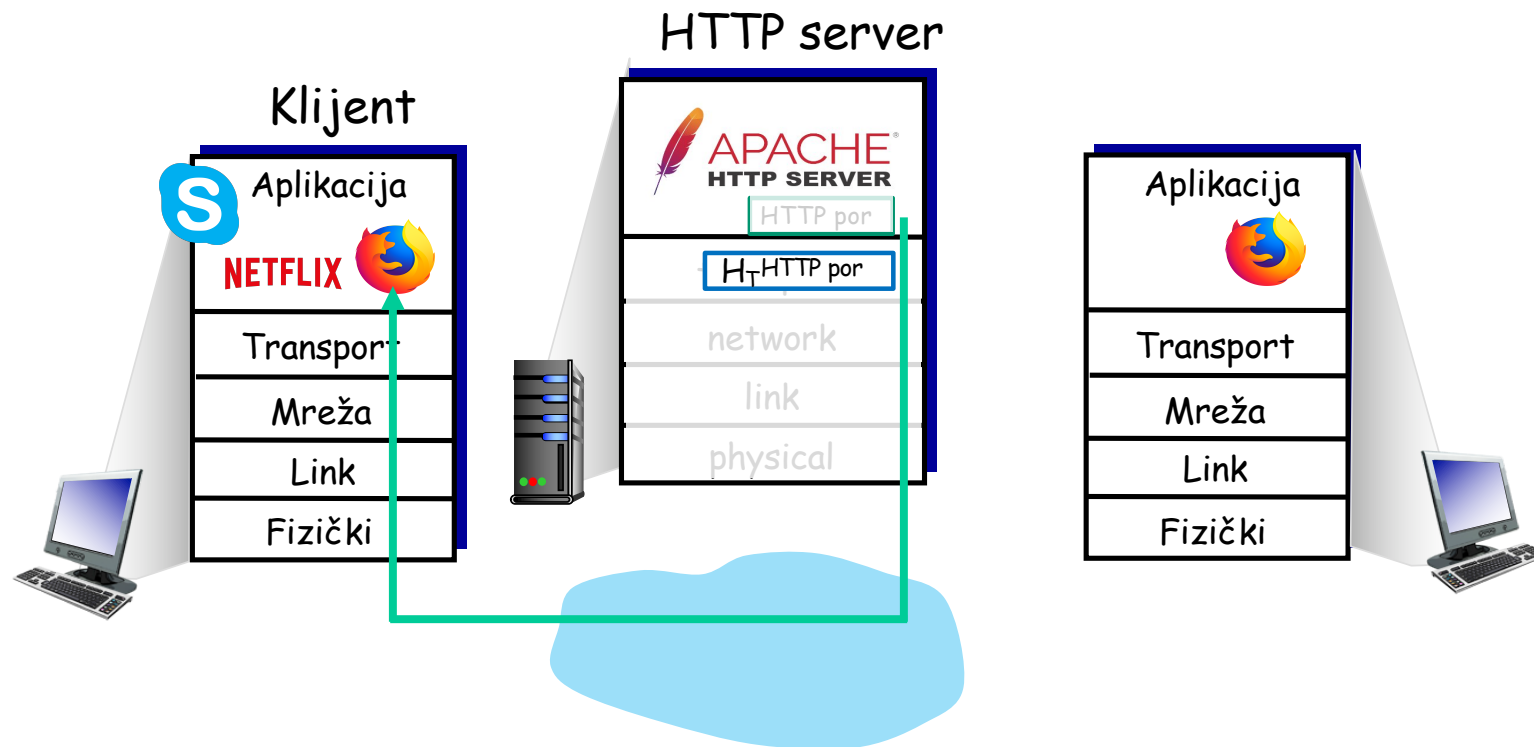
3.5 Konektivni transport: TCP

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

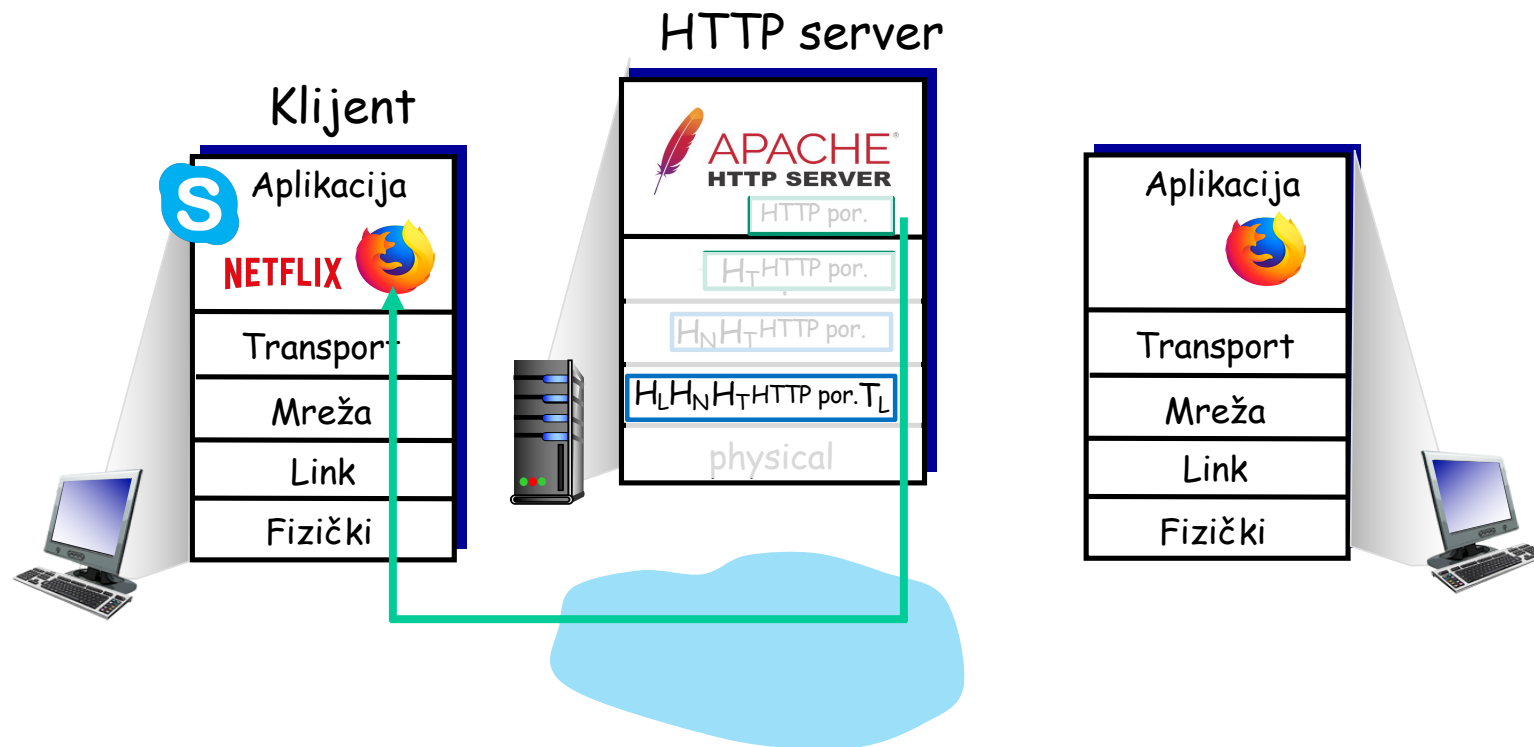
Mutlipleksiranje i demultipleksiranje



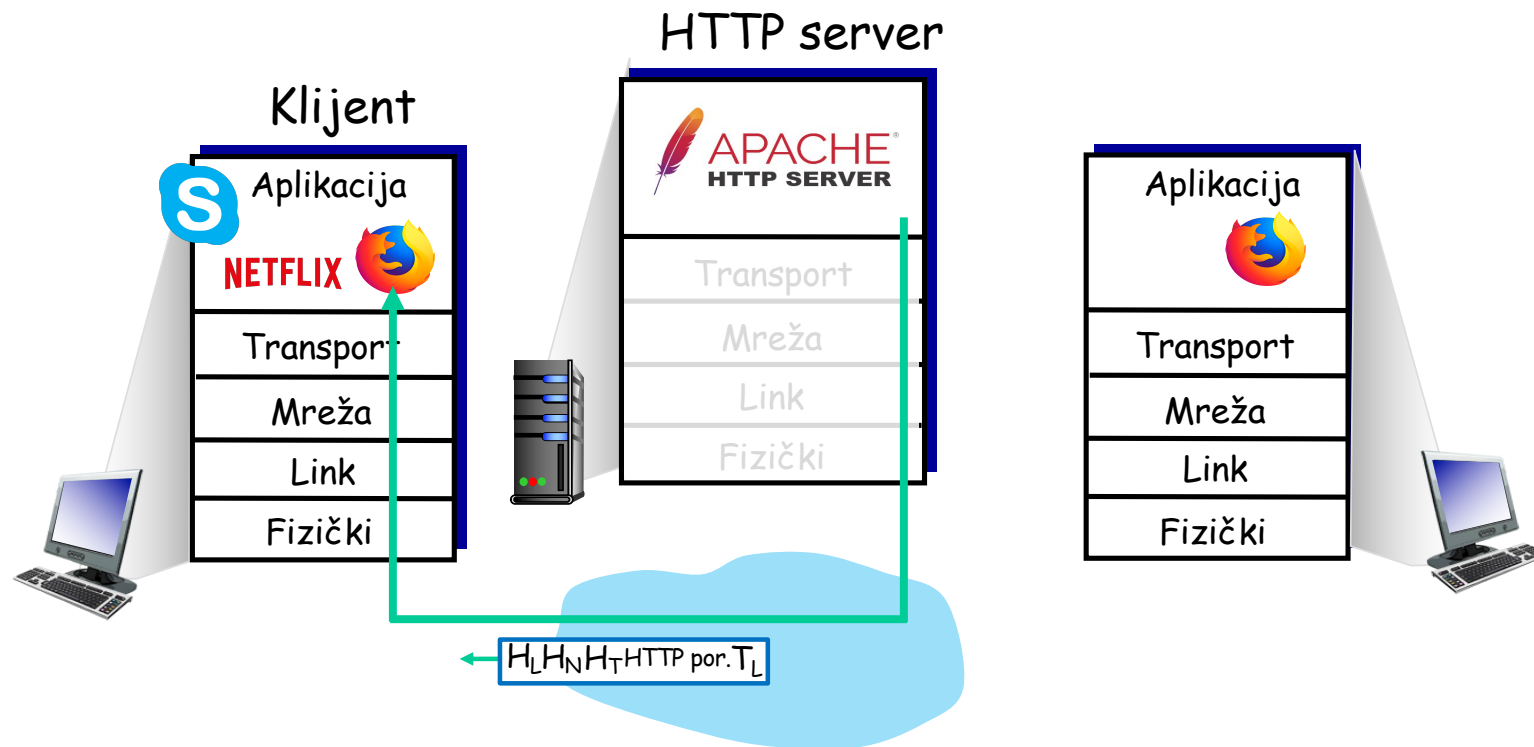
Mutlipleksiranje i demultipleksiranje



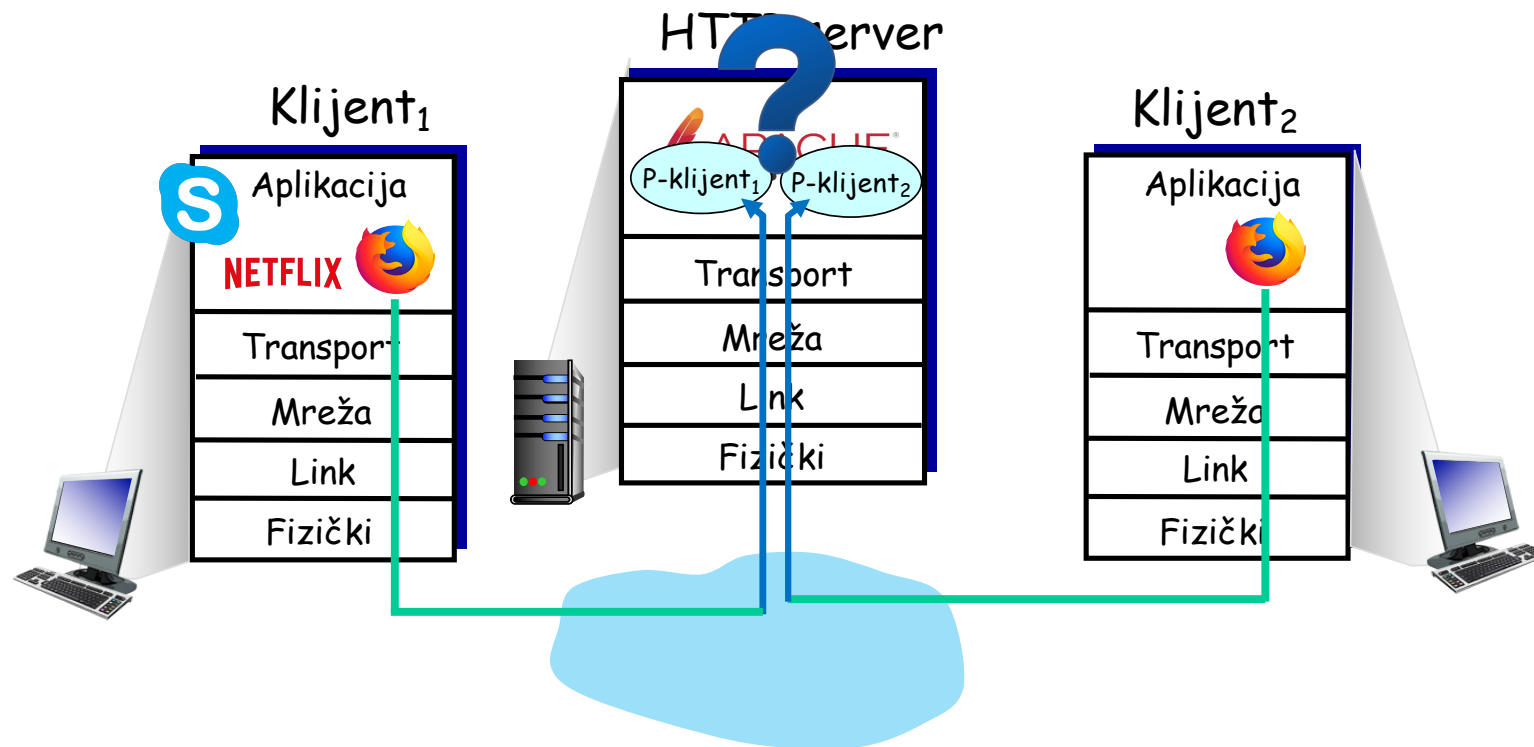
Mutlipleksiranje i demultipleksiranje



Multiplexiranje i demultiplexiranje



Multiplexiranje i demultiplexiranje



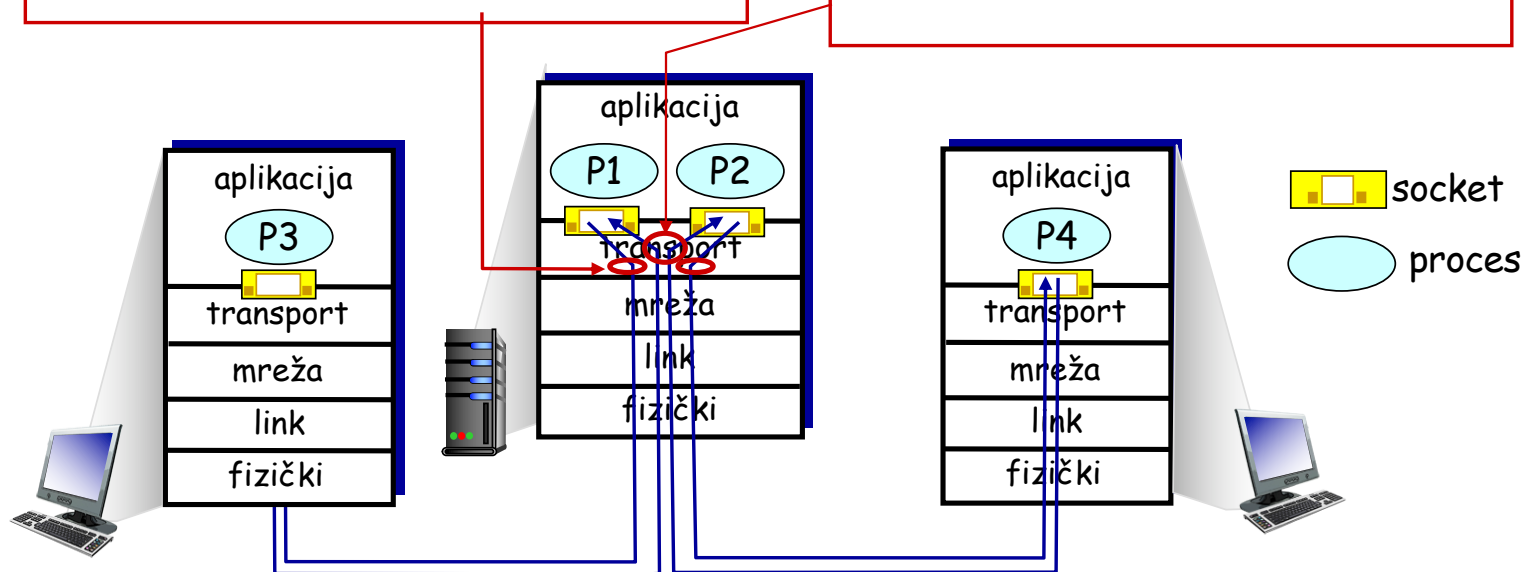
Multiplexiranje i demultiplexiranje

Multiplexiranje na predaji:

Manipulisanje podacima iz više socketa, dodavanje transportnog zaglavlja (koristi se za demultiplexiranje)

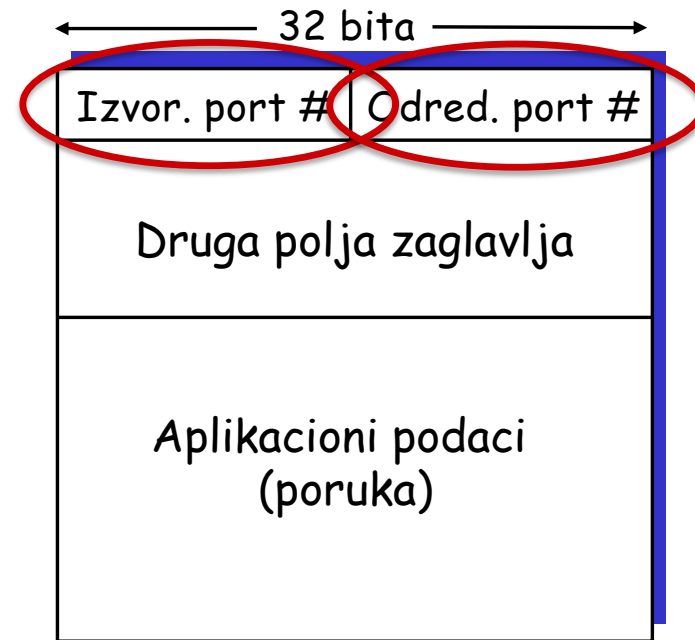
Demultiplexiranje na prijemu:

Koristi zaglavlje za predaju primljenih segmenata pravom socketu



Kako funkcioniše demultipleksiranje?

- ❑ host prima IP datagrame
 - Svaki datagram ima izvorišnu IP adresu i 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)
- ❑ host koristi IP adrese i brojeve portova da usmjeri segment na odgovarajući socket

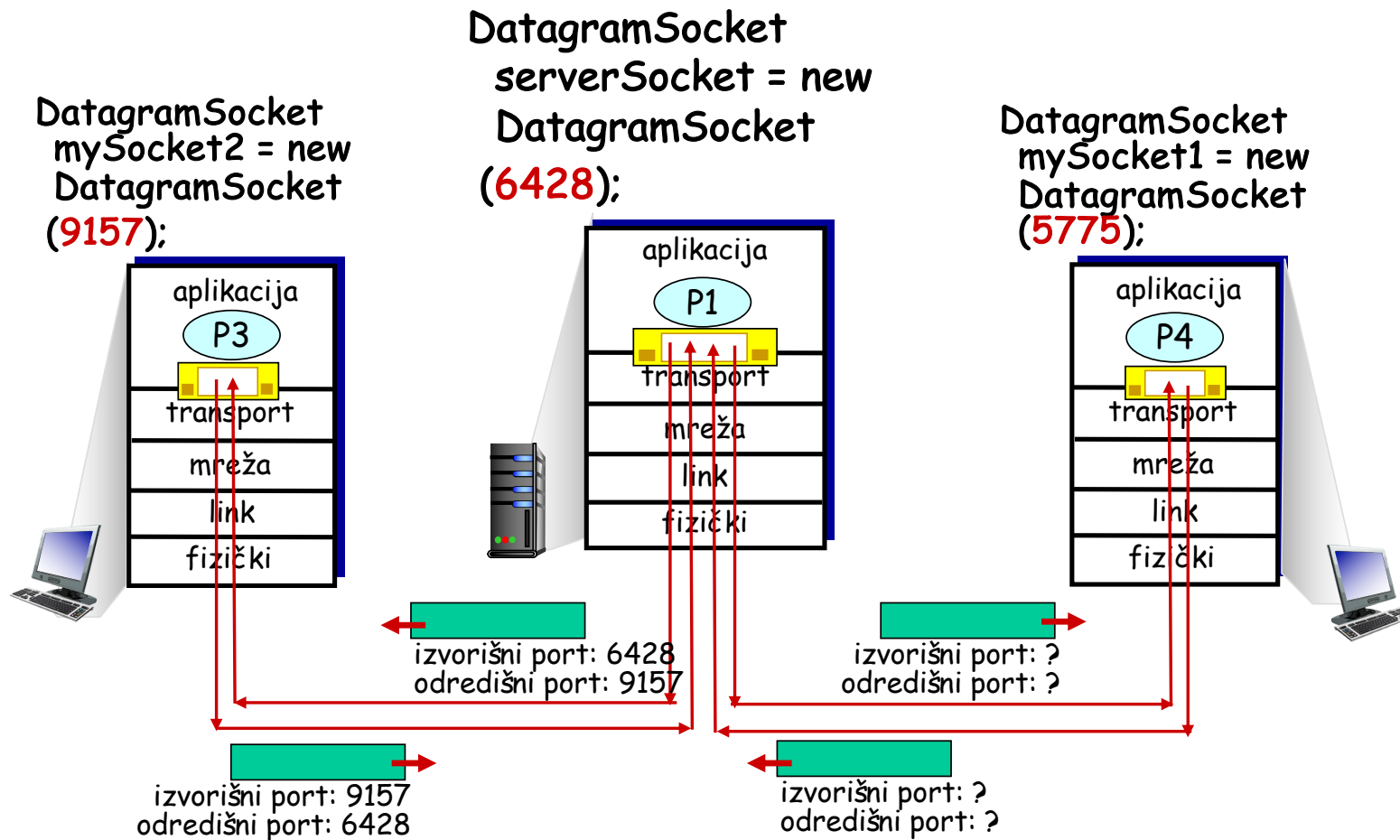


TCP/UDP format segmenta

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 socketu dodjeljuje ne "dobro poznate" brojeve portova (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 istim destinacionim brojevima portova, ali različitim izvorišnim IP adresama i/ili izvorišnim brojevima portova se usmjeravaju na isti socket

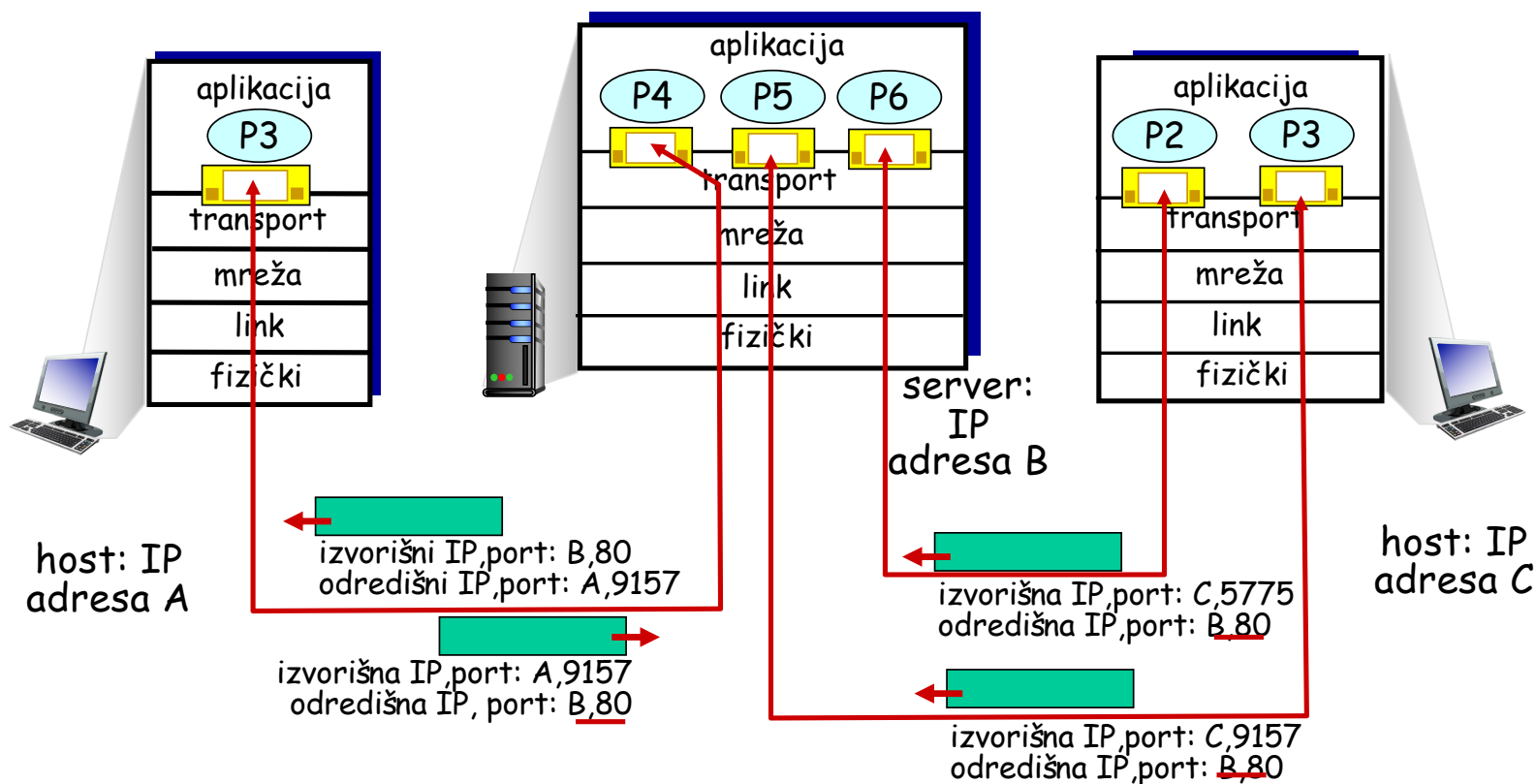
Nekonektivno demultipleksiranje



Konektivno demultipleksiranje

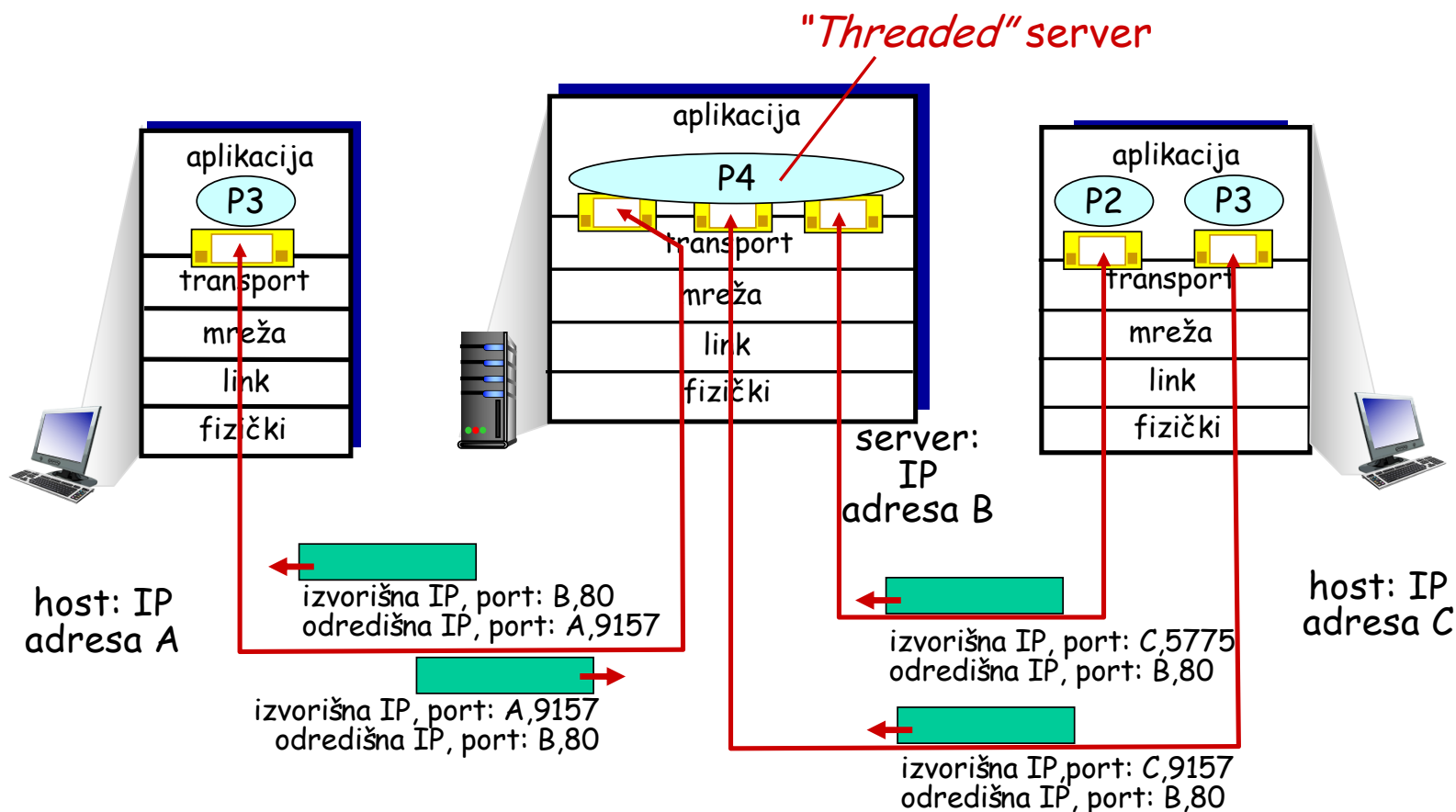
- ❑ TCP socket identifikuju 4 parametra:
 - Izvorišna IP adresa
 - Izvorišni broj porta
 - Odredišna IP adresa
 - Odredišni broj porta
- ❑ Prijemni host koristi sve četiri vrijednosti za usmjeravanje segmenta na odgovarajući socket
- ❑ Server može podržavati više simultanih TCP socket-a:
 - svaki socket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

Konektivno demultipleksiranje



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

Konektivno demultipleksiranje



3. Nivo transporta

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

3.4 Principi pouzdanog prenosa podataka

3.5 Konektivni transport: TCP

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

UDP: User Datagram Protocol [RFC 768]

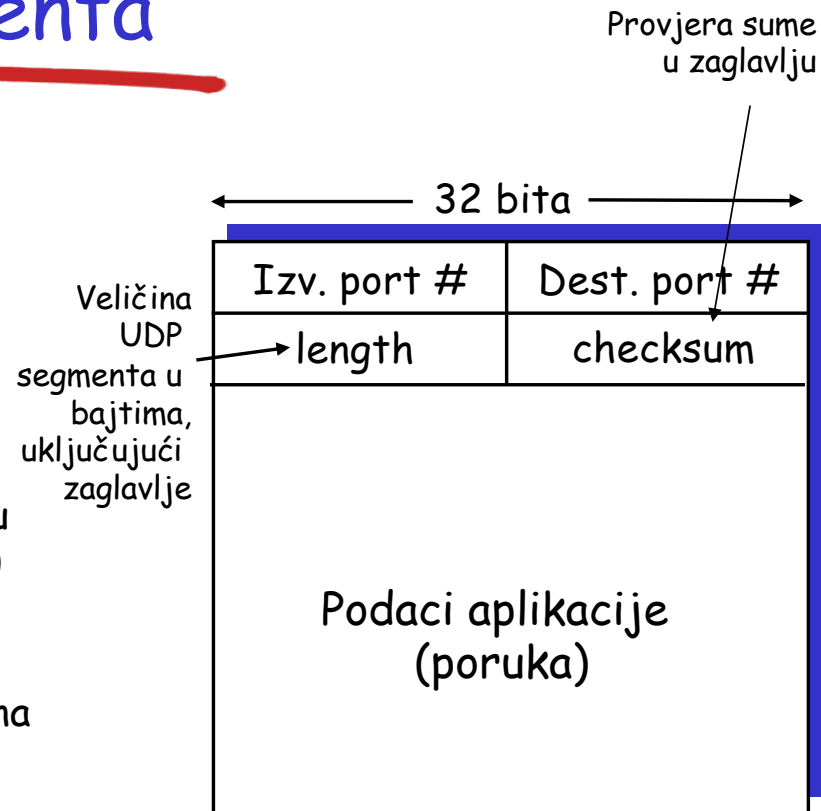
- ❑ Nema poboljšanja koja se nude Internet protokolu
- ❑ “best effort” servis, UDP segmenti mogu biti:
 - izgubljeni
 - predani neredosledno
- ❑ *nekonektivni*:
 - nema uspostavljanja veze (*handshaking*) između UDP pošiljaoca i prijemnika
 - svaki UDP segment se tretira odvojeno od drugih

Zašto onda UDP?

- ❑ Nema uspostavljanja veze (koja povećava kašnjenje)
- ❑ Jednostavnije jer se ne vodi računa o stanju veze
- ❑ manje zaglavlje segmenta (8B u odnosu na 20B kod TCP protokola)
- ❑ nema kontrole zagušenja: UDP može slati podatke onom brzinom kojom to aplikacija želi

UDP: zaglavlje segmenta

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



Format UDP segmenta
RFC 768

UDP checksum-a

Cilj: detekcija greške u prenošenom segmentu

Razlog: nema garancije da je kontrola greške primijenjena na svim linkovima preko kojih se segment prenosi. Dodatno, greška može nastupiti i u nekom od rutera.

Pošiljalac:

- ❑ Tretira sadržaj segmenta (uključujući zaglavlje) kao sekvence 16-bitnih brojeva
- ❑ Izračunava se 1. komplement sume 16-bitnih brojeva (*checksum*)
- ❑ Pošiljac postavlja vrijednost *checksum*-e u odgovarajuće polje UDP segmenta

Prijemnik:

- ❑ Sekvence 16-bitnih brojeva zaglavlja (uključujući polje *checksum*) se sumiraju i provjerava se da li je rezultat broj koji sadrži 16 jedinica:
 - NE - detektovana greška
 - DA - nema greške. ????????

Nema oporavka od greške! Segment se odbacuje ili se predaje aplikaciji uz upozorenje!

Primjer Internet Checksum-e

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
prenos	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
suma	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Kada se sabiraju brojevi, prenos sa najznačajnijeg bita se dodaje rezultatu

3. Nivo transporta

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

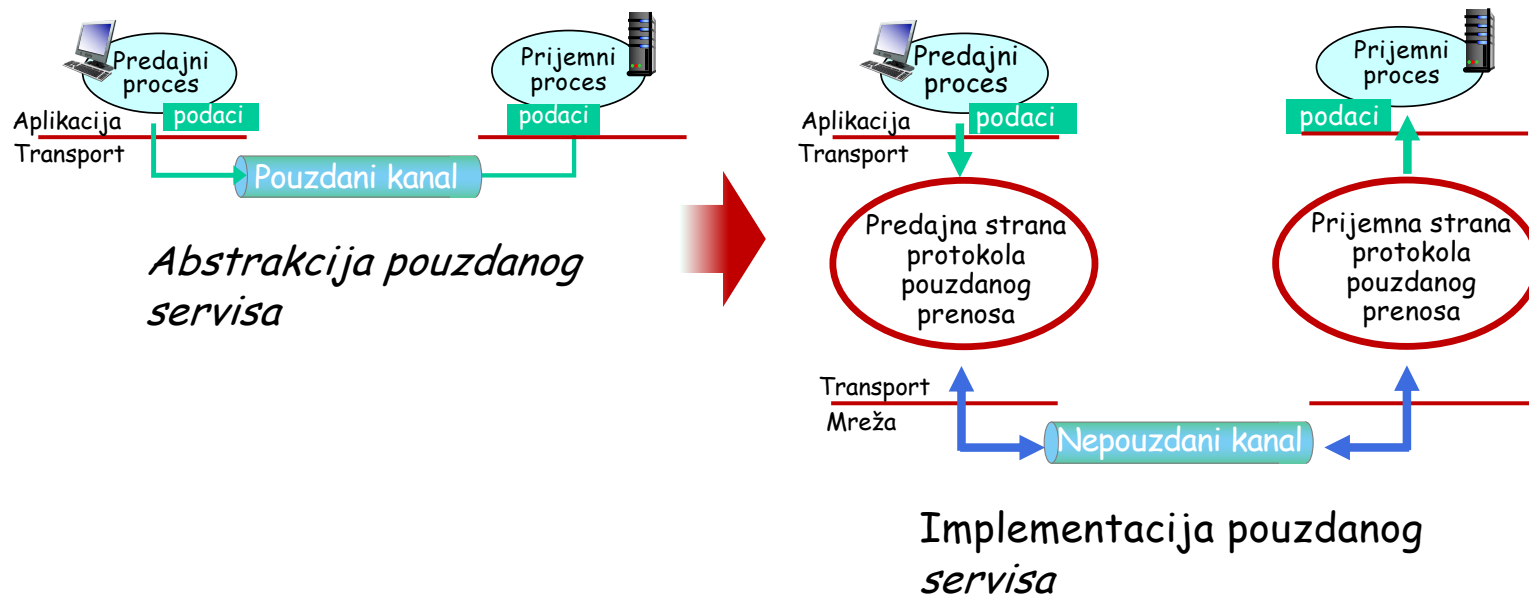
3.4 Principi pouzdanog prenosa podataka

3.5 Konektivni transport: TCP

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

Opšti principi pouzdanog prenosa podataka

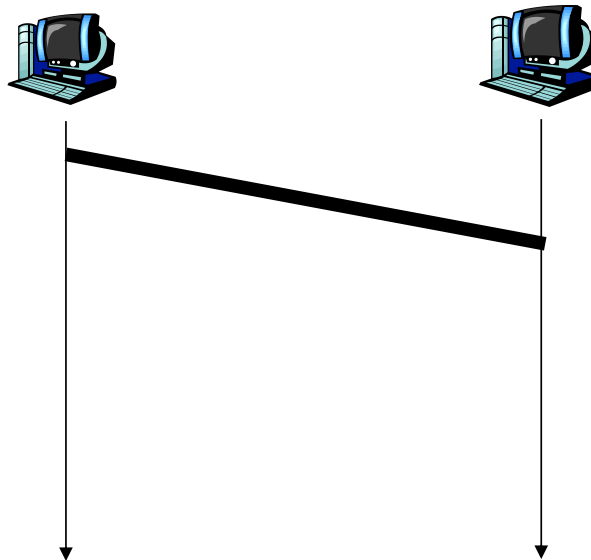
- Pouzdani prenos je važan na nivoima **aplikacije, transporta, linka**
- Jedna od top-10 karakteristika mreže!



- Karakteristike nepouzdanog kanala će odrediti kompleksnost protokola zapouzdanog prenosa podataka (rdt)

Pouzdaní prenos preko pouzdanog kanala

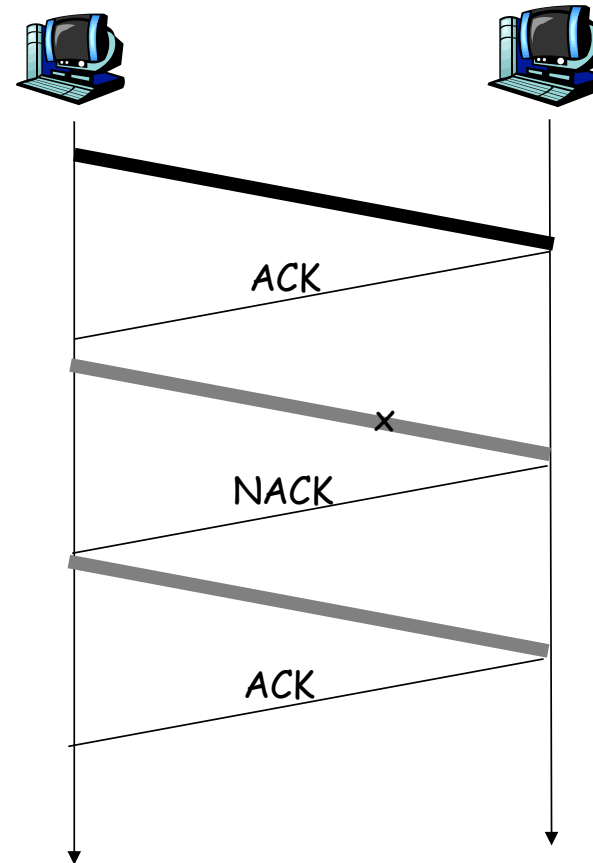
- Kanal je pouzdan u potpunosti
 - Nema greške po bitu
 - Nema gubitka paketa



- Nema potrebe za protokolom pouzdanog prenosa!

Kanal sa greškom (ali nema gubitka paketa)

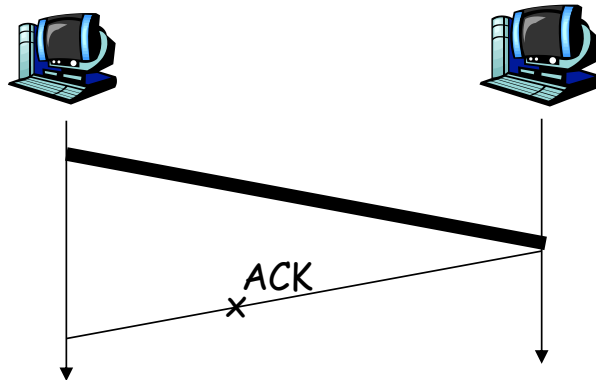
- ❑ Kanal može zamijeniti vrijednosti bita u paketu
- ❑ Potrebno je detektovati grešku na prijemnoj strani. Kako?
- ❑ Prijemna strana o tome mora obavijestiti predajnu stranu potvrdom uspjehnog (ACK) ili neuspješnog prijema (NACK)
- ❑ Ako prijemna strana primi
 - ACK - šalje nove podatke
 - NACK - ponovo šalje prethodni paket (retransmisija)
- ❑ ARQ (*Automatic Repeat reQuest*)



Prethodno rješenje ima fatalan problem!

Šta se dešava kada su ACK/NAK oštećene?

- Pošiljalac ne zna šta se dešava na prijemu!
- Retransmisija je besmislena: moguće je dupliranje paketa na prijemnoj strani

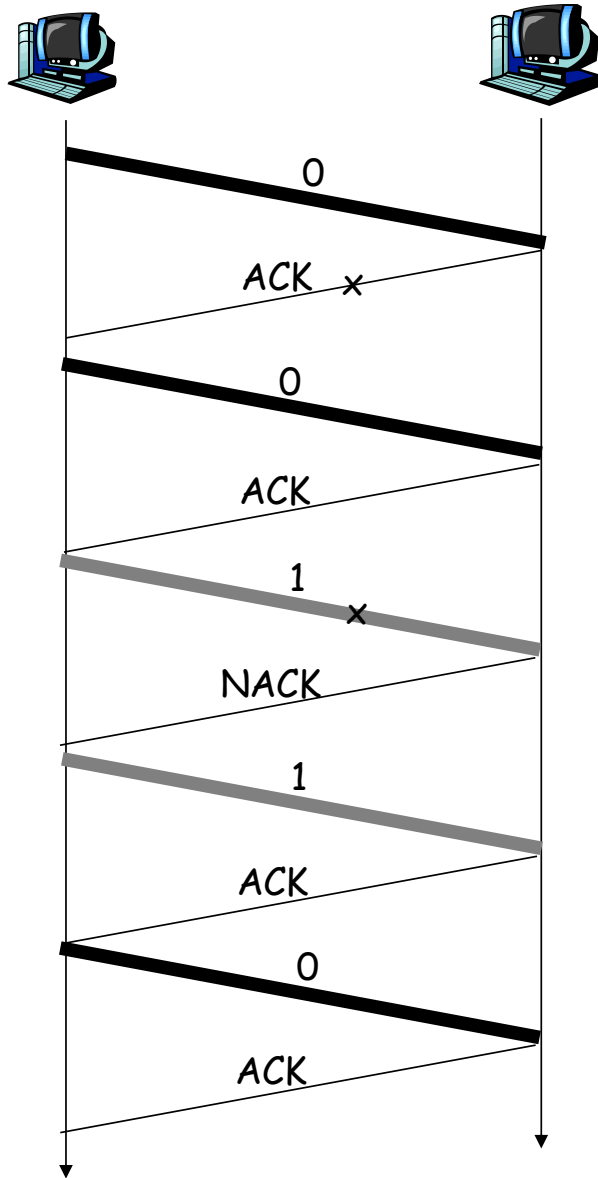


Rješavanje problema duplikata:

- Pošiljalac dodaje svakom paketu broj u sekvenci
- Pošiljalac ponovo šalje posmatrani paket ako je ACK/NAK oštećen
- Prijemnik odbacuje duple pakete
- U ACK/NAK nema broja u sekvenci paketa koji se potvrđuje jer nema gubitka paketa, pa se potvrda odnosi na poslednji poslani paket.

STOP & WAIT

Pošiljac šalje jedan paket, a zatim čeka na odgovor



Stop & Wait u kanalu bez gubitaka

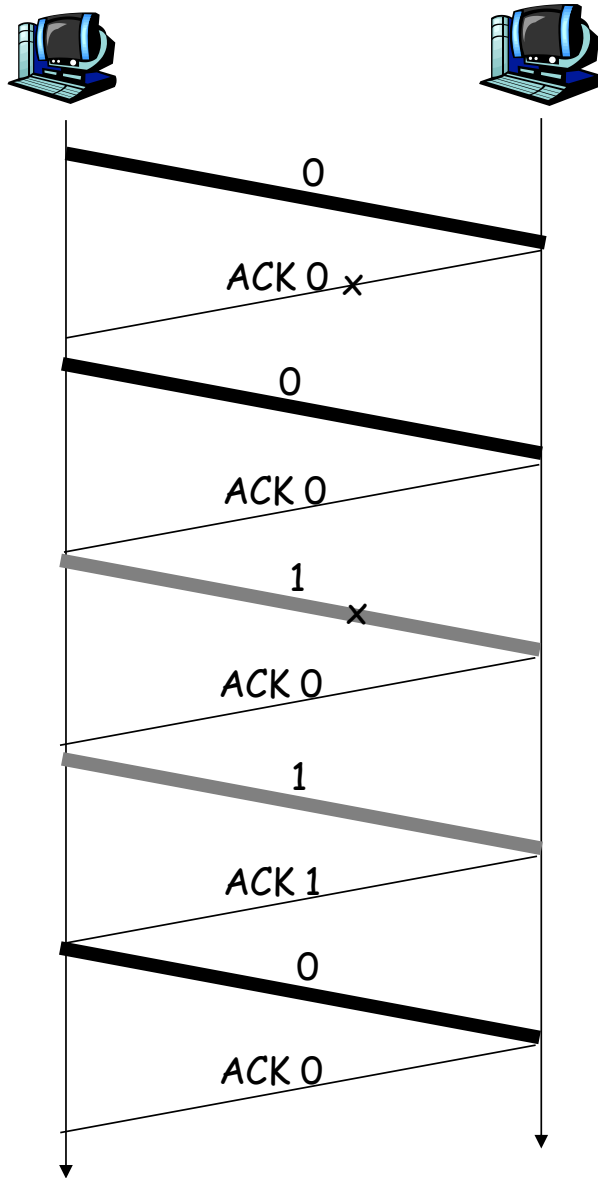
Stop & Wait u kanalu bez gubitaka

Pošiljalac:

- ❑ Dodaje broj u sekvenci paketu
- ❑ Dva broja (0,1) su dovoljna.
Zašto?
- ❑ Mora provjeriti da li je primljeni ACK/NAK oštećen

Prijemnik:

- ❑ Mora provjeriti da li je primljeni paket duplikat
 - stanje indicira da li je 0 ili 1 očekivani broj u sekvenci paketa
- ❑ Napomena: prijemnik ne može znati da li je poslednji ACK/NAK primljen ispravan od strane pošiljaoca



Stop & Wait u kanalu bez gubitaka i bez NAK

- Iste funkcionalnosti kao u prethodnom slučaju, korišćenjem samo ACK
- umjesto NAK, prijemnik šalje ACK za poslednji paket koji je primljen ispravno
 - Prijemnik mora *eksplicitno* unijeti broj u sekvenci paketa čiji se uspješan prijem potvrđuje
- Dvostruki ACK za isti paket na strani pošiljaoca rezultira istom akcijom kao: *ponovo šalji posmatrani paket*

Stop & wait u kanalu sa greškom i gubicima

Nova pretpostavka: kanal izaziva i gubitak paketa (podataka ili potvrda)

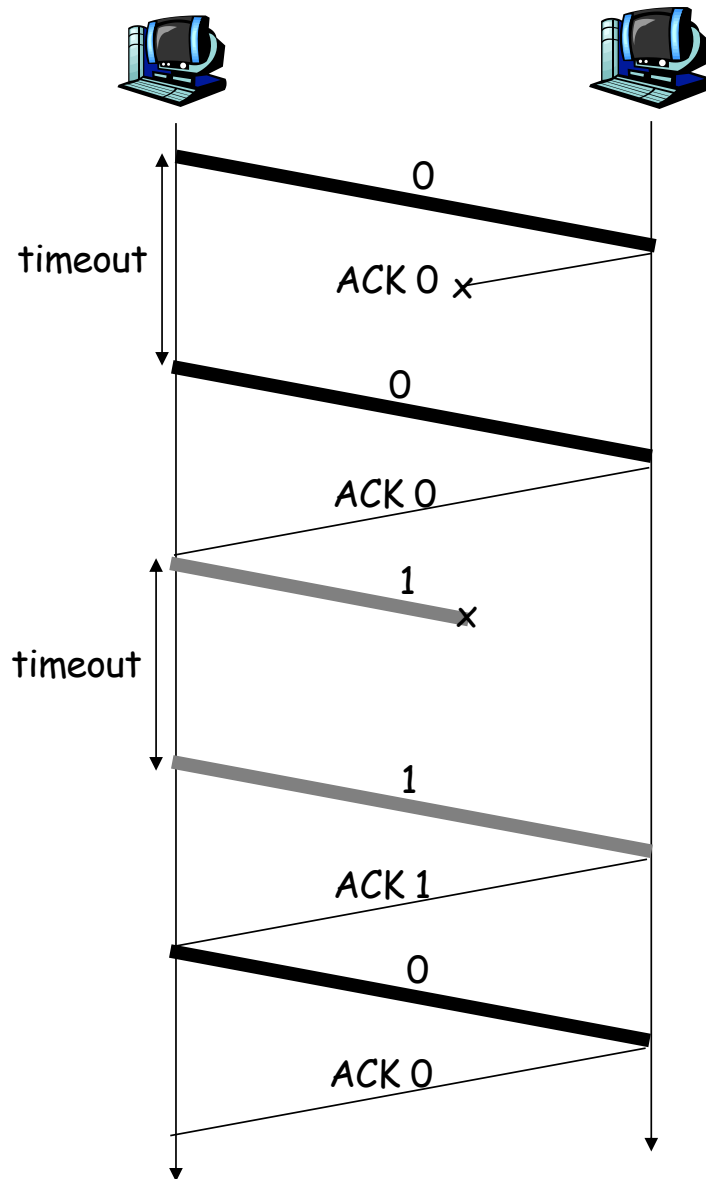
- checksum, broj u sekvenci, ACK, retransmisije su od pomoći, ali ne dovoljno.

Kako se izboriti sa gubicima?

- Pošiljalac čeka dok se određeni podaci ili ACK izgube, zatim obavlja retransmisiju.
- Koliko je minimalno vrijeme čekanja?
- Koliko je maksimalno vrijeme čekanja?
- Nedostaci?

Pošiljalac može da čeka “razumno” vrijeme za ACK

- Retransmisija se obavlja ako se ACK ne primi u tom vremenu
- Ako paket (ili ACK) samo zakasni (nije izgubljen):
 - Retransmisija će biti duplirana, ali korišćenje broja u sekvenci će to odraditi
 - Prijemnik mora definisati broj u sekvenci paketa čiji je prijem već potvrđen
- Zahtijeva timer

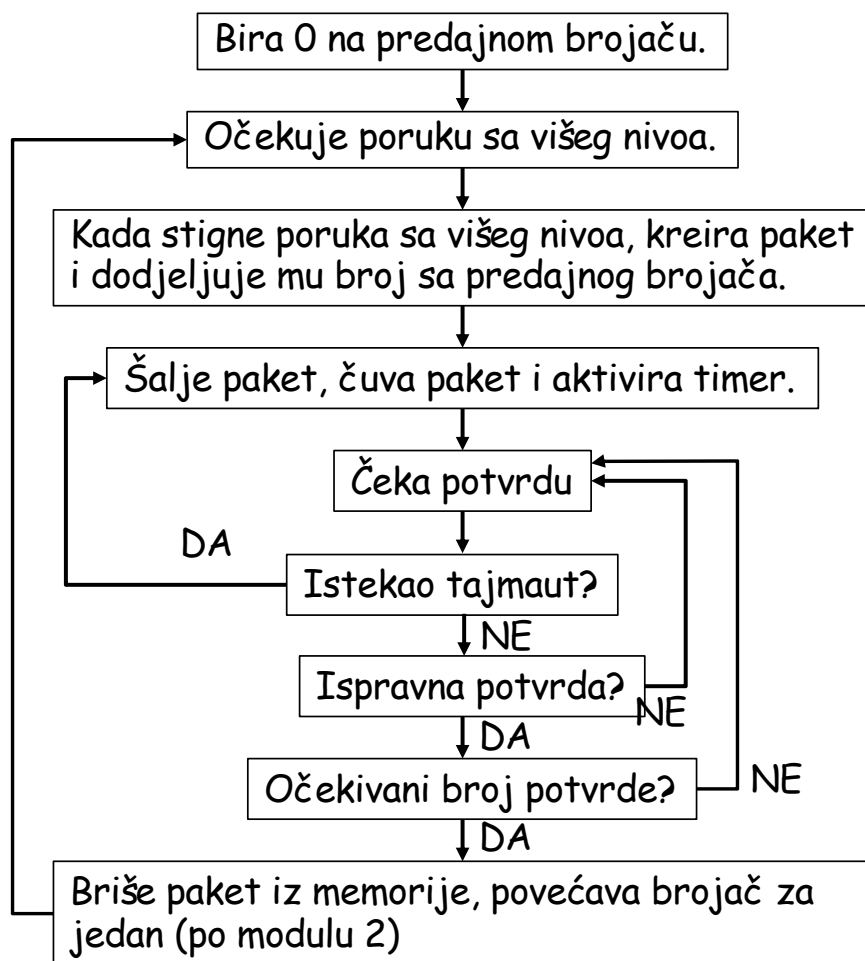


Stop & Wait u kanalu sa gubicima

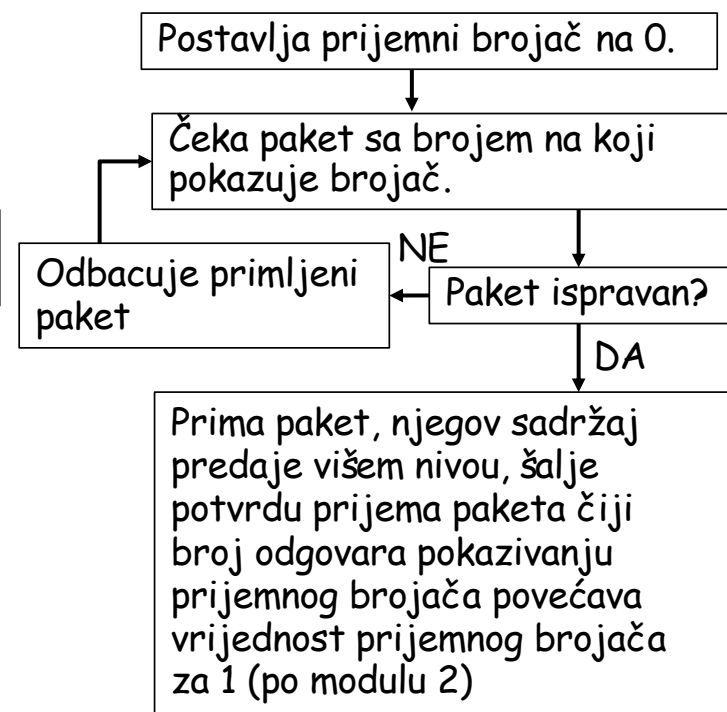
- Iste funkcionalnosti kao u prethodnom slučaju, korišćenjem samo ACK
- umjesto NAK, prijemnik šalje ACK za poslednji ispravno primljeni paket
 - Prijemnik mora *eksplicitno* unijeti broj u sekvenci paketa čiji se uspješan prijem potvrđuje
- Dvostruki ACK za isti paket na strani pošiljaoca rezultira istom akcijom kao "*ponovo šalji posmatrani paket*"

Stop & Wait

Pošiljalac:



Prijemnik:



STOP & WAIT performanse

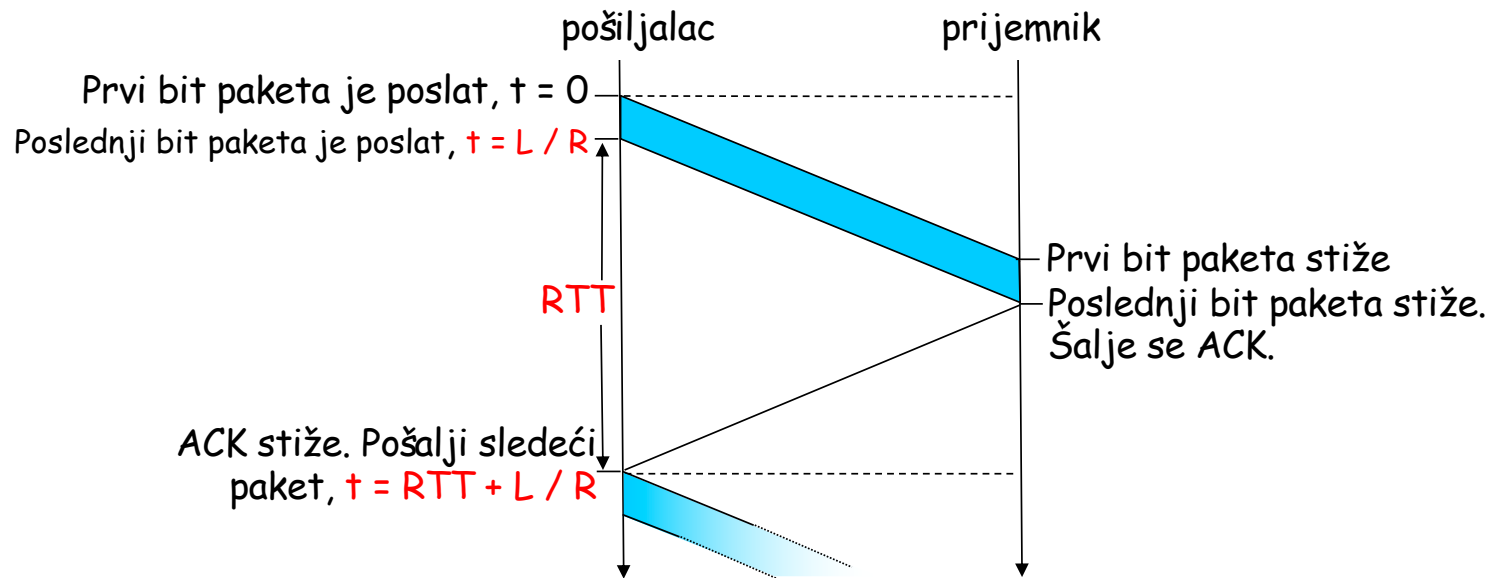
- S&W funkcioniše, ali ima loše performanse
- primjer: 1 Gb/s link, 15 ms vrijeme prenosa od kraja do kraja, veličina paketa 1000 B :

$$T_{\text{prenosa}} = \frac{L \text{ (veličina paketa u bitima)}}{R \text{ (kapacitet linka [b/s])}} = \frac{8 \text{ kb/paketu}}{10^9 \text{ b/s}} = 8 \mu\text{s}$$

$$U_{\text{Pošilj.}} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$

- $U_{\text{pošiljalac}}$: **iskorišćenje** - dio vremena u kome je pošiljalac zauzet
- Pošiljalac šalje 1000B paket svakih 30,008 ms -> 267 kb/s bez obzira što je kapacitet linka 1Gb/s
- Primjer da mrežna funkcija ograničava propusnost mreže!
- U praksi je situacija još gora jer je napravljeno nekoliko zanemarivanja!

STOP & WAIT performanse

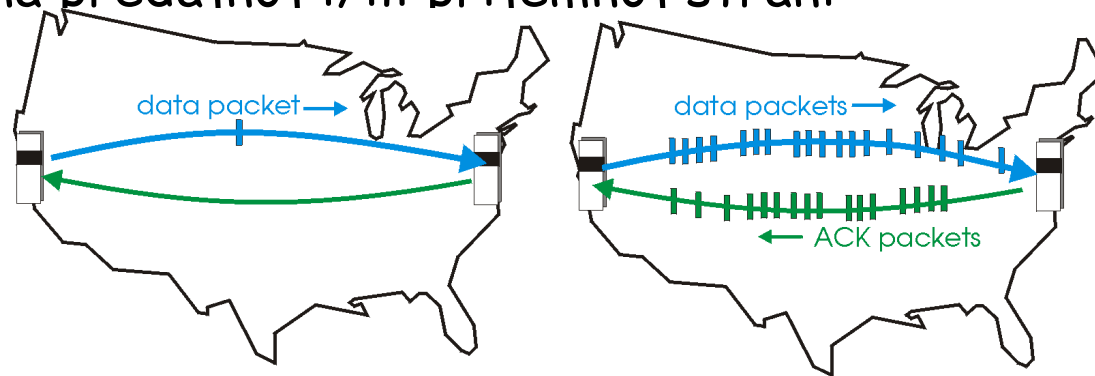


$$U_{\text{Pošilj.}} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$

"Pipelined" protokoli

"Pipelining": pošiljalac dozvoljava istovremeni prenos više paketa čiji prijem nije potvrđen

- Opseg brojeva u sekvenci mora biti proširen
- Baferovanje na predainoi i/ili prijemnoi strani

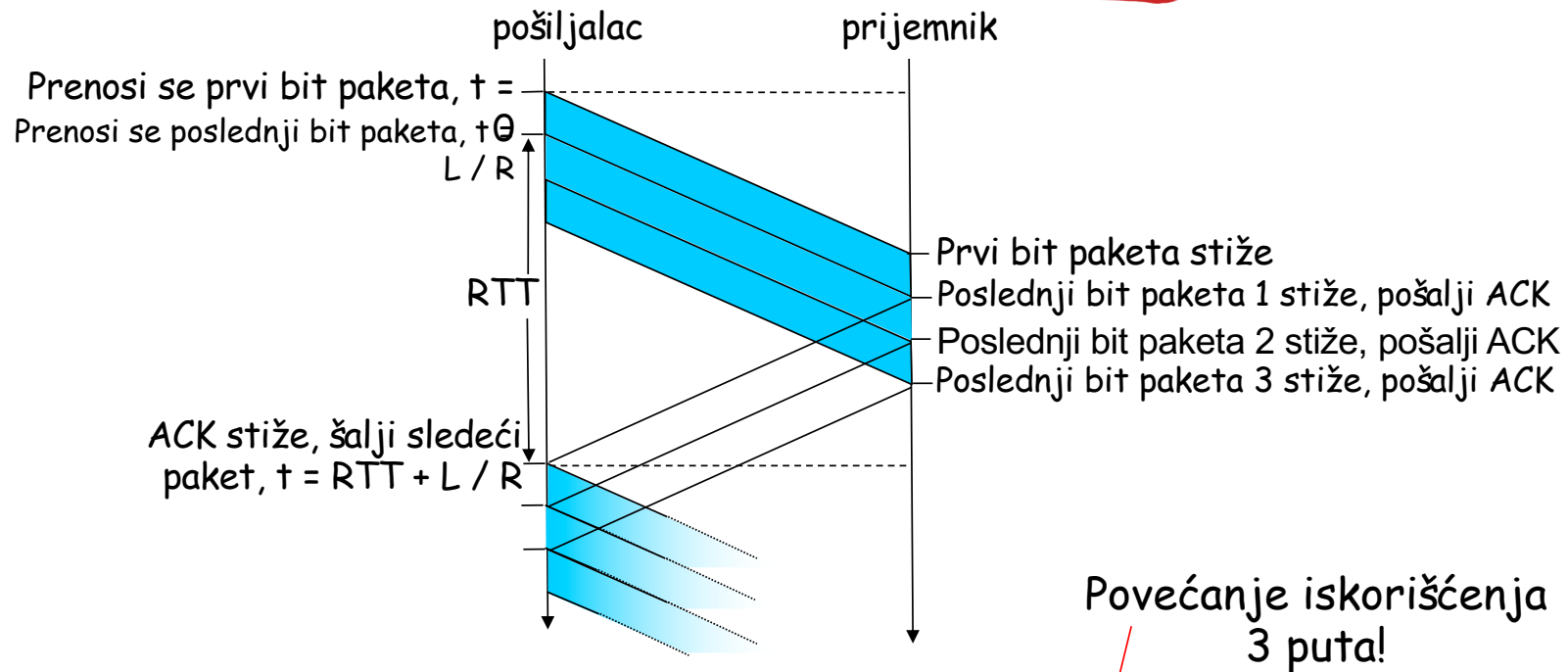


a) Stop and wait protokol

b) Pipeline protokol

- Postoje dvije forme ovog protokola: "*go-Back-N*", "*selective repeat*"

"Pipelining" povećava iskorišćenje



$$U_{\text{Pošilj.}} = \frac{3 * L / R}{RTT + L / R} = \frac{0,024}{30,008} = 0,0008$$

Pipelined protokoli

Go-back-N:

- ❑ Pošiljalac može imati do N nepotvrđenih poslatih segmenata
- ❑ Prijemnik šalje samo *kumulativne potvrde*
 - ❑ Ne potvrđuje segmente ako se jave "praznine"
- ❑ Pošiljalac ima timer za najstariji nepotvrđeni paket
 - ❑ Kada timer istekne ponovo se šalju svi nepotvrđeni segmenti

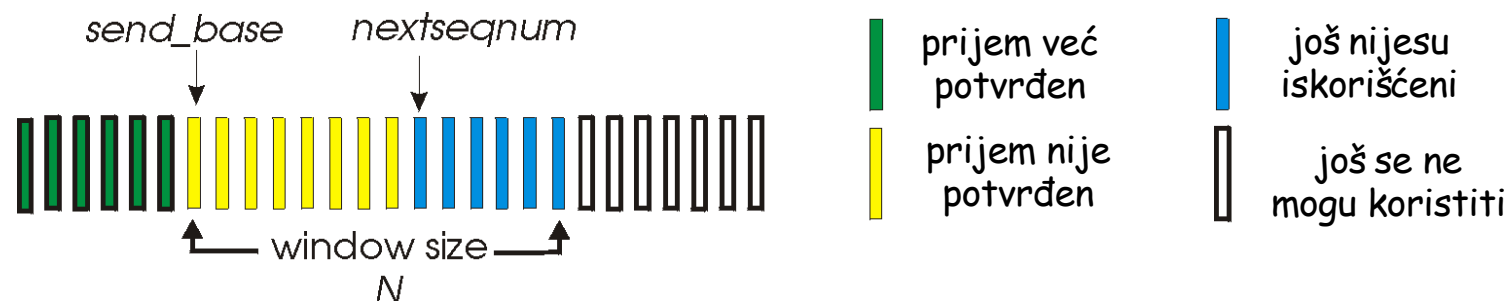
Selective Repeat:

- ❑ Pošiljalac može imati do N nepotvrđenih poslatih segmenata
- ❑ Prijemnik šalje *individualne potvrde* za svaki paket
- ❑ Predajnik ima tajmer za svaki nepotvrđeni segment
 - ❑ Kada timer istekne ponovo se šalje samo taj segment

Go-Back-N (sliding window)

Pošiljalac:

- k-bitni dugačak broj u sekvenci u zaglavlju paketa znači da se može poslati $N=2^k$ nepotvrđenih paketa
- “prozor” veličine N susjednih nepotvrđenih paketa je dozvoljen
- Zašto ograničavati N?



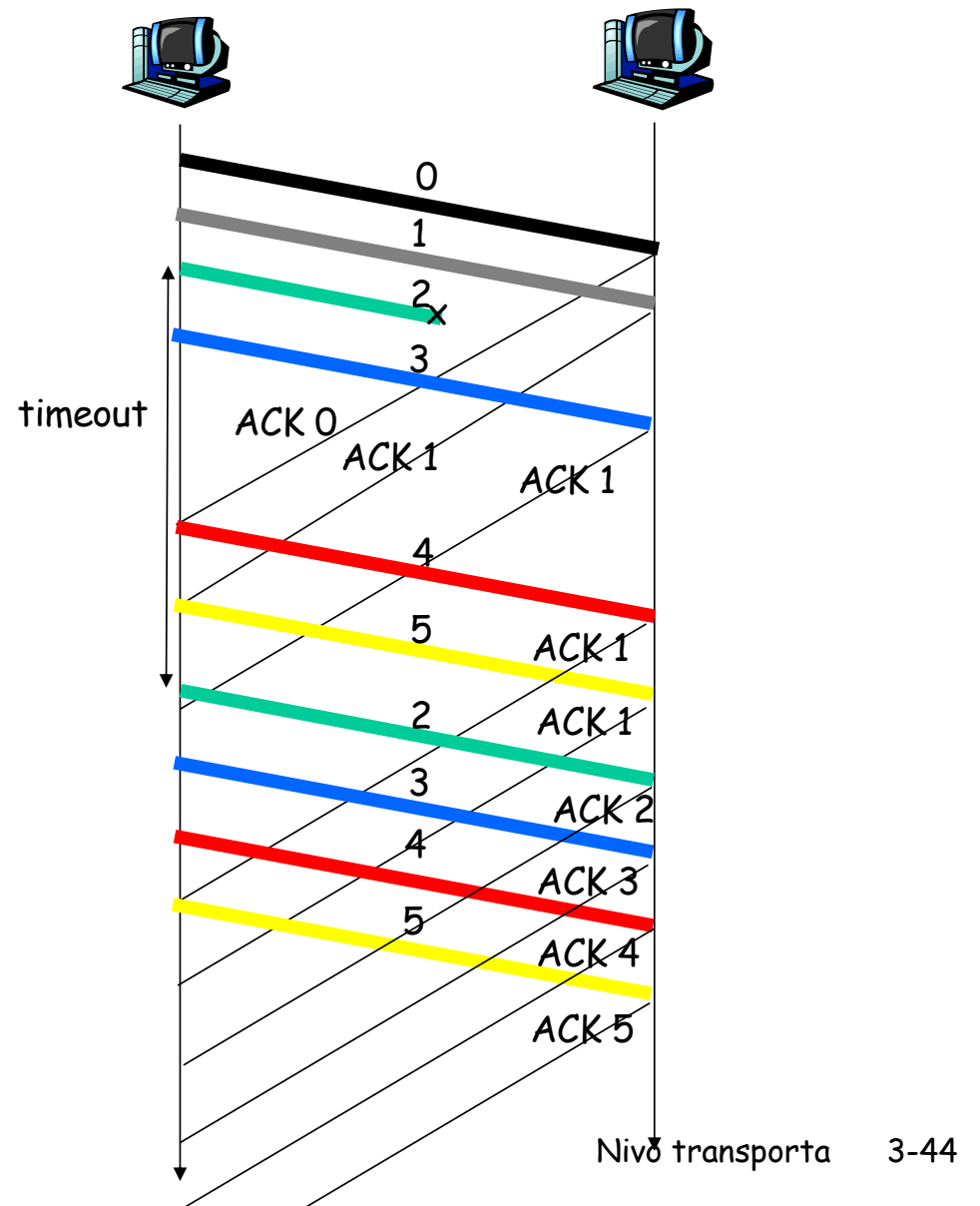
- Broj u sekvenci se upisuje u polje zaglavlja veličine k bita ($0, 2^k-1$). Kod TCP ($k=32$) se ne broje segmenti, već bajti u streamu.
- ACK (n): ACK sve pakete, uključujući n-ti u sekvenci - “kumulativni ACK”
 - Mogu se pojaviti dupli ACK-ovi (vidi prijemnik)

GBN

- ❑ timer se inicijalizuje za "najstariji" segment i vezuje za svaki paket čiji prijem još nije potvrđen
- ❑ *timeout(n)*: retransmisija paketa n i svih paketa čiji je broj u sekvenci veći od n, u skladu sa veličinom prozora
- ❑ uvijek se šalje ACK za korektno primljen paket sa najvećim brojem u sekvenci uz poštovanje *redosleda*
 - Može generisati duple ACK-ove
 - Treba da zapamti samo broj očekivanog paketa
- ❑ "out-of-order" paket:
 - odbacuje -> nema baferovanja na prijemu! Zašto?
 - Re-ACK paket sa najvećim brojem u sekvenci

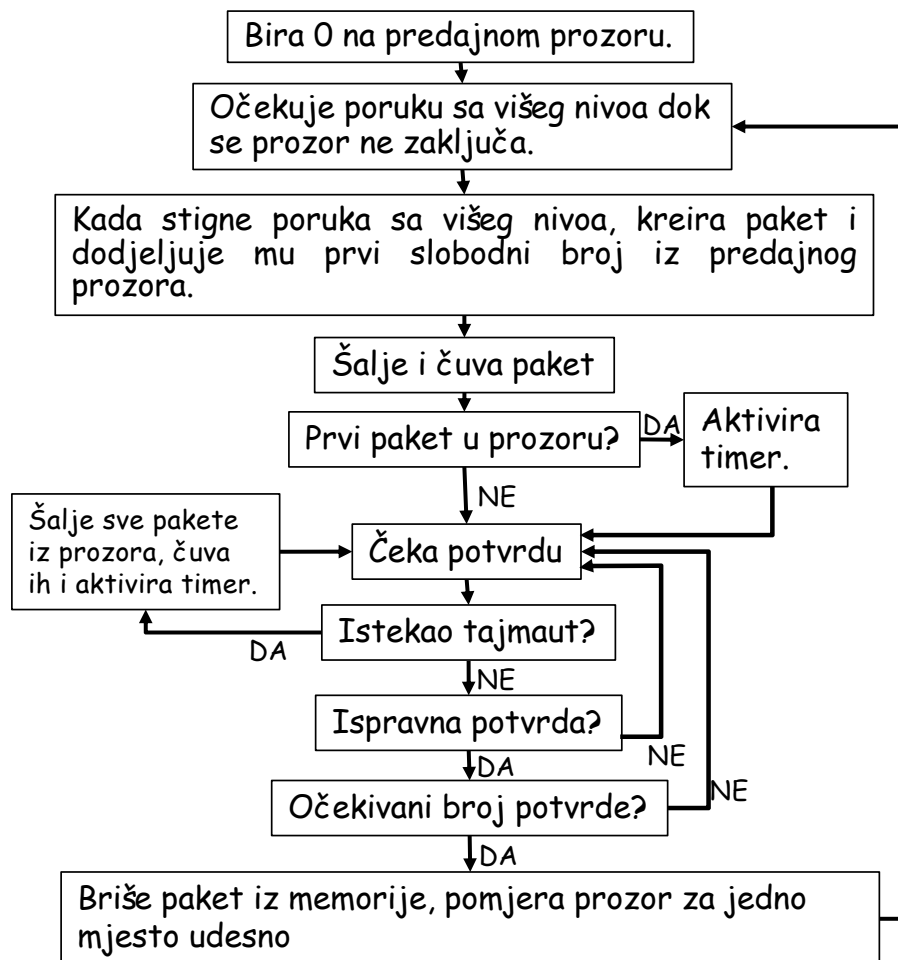
GBN u akciji

Predajni prozor(N=4)

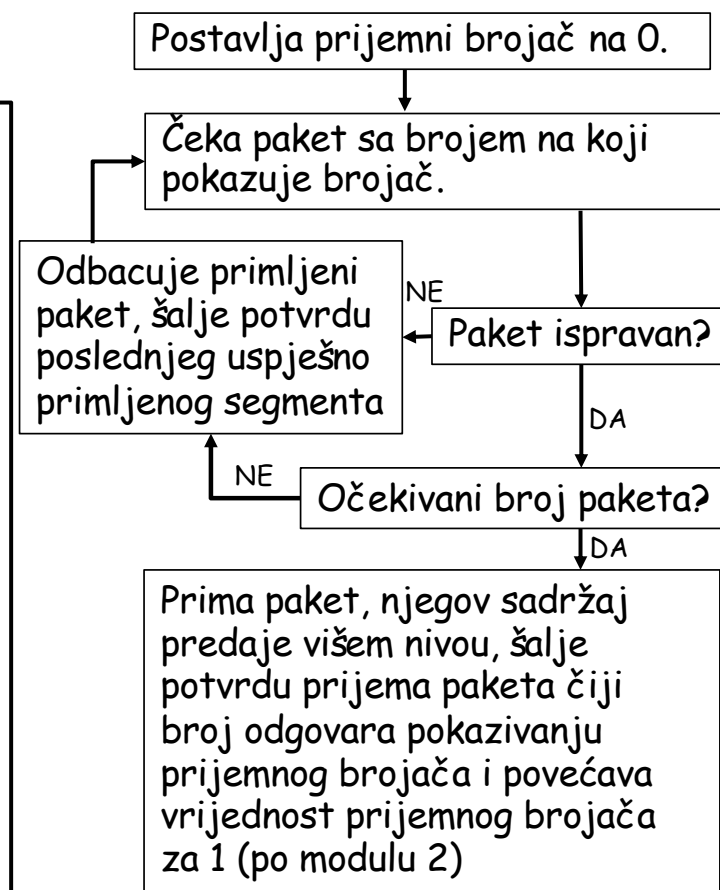


GBN

Pošiljalac:



Prijemnik:



Go-Back-N: problemi

- ❑ Dozvoljava pošiljaocu da ispuni link sa paketima, čime se uklanja problem lošeg iskorišćenja kanala.
- ❑ Sa druge strane, kada su veličina prozora i proizvod brzine prenosa i kašnjenja veliki mnogo paketa može biti na linku. U slučaju gubitka jednog paketa mnogi paketi moraju biti ponovo poslati.
- ❑ Iz tog razloga se koriste SELECTIVE REPEAT protokoli, koji kao što im ime kaže, omogućavaju izbor paketa koji će biti ponovo poslati.

"Selective Repeat"

- ❑ Prijemnik *pojedinačno* potvrđuje sve ispravno primljene pakete
 - baferuje pakete, ako je to potrebno, za eventualnu redoslednu predaju nivou iznad sebe
- ❑ Pošiljalac ponovo šalje samo pakete za koje ACK nije primljen
 - Pošiljalac ima tajmer za svaki paket čiji prijem nije potvrđen
- ❑ Prozor pošiljaoca
 - N uzastopnih brojeva u sekvenci
 - Ponovo ograničava broj poslatih paketa, čiji prijem nije potvrđen

"Selective repeat"

Pošiljalac

Podaci odozgo:

- Ako je sledeći broj u sekvenci u prozoru dostupan, šalji paket

timeout(n):

- Ponovo šalji paket n, restartuj tajmer

ACK(n) u [sendbase, sendbase+N]:

- Markiraj paket n kao da je primljen
- Ako je n najmanji nepotvrđeni paket, proširi osnovu prozora na bazi narednog najmanjeg broja nepotvrđenog paketa

Prijemnik

paket n u [rcvbase, rcvbase+N-1]

- Pošalji ACK(n)
- out-of-order: baferuj
- in-order: predaj (takođe baferuj, predaj u in-order), povećavaj prozor na sledeći paket koji još nije primljen

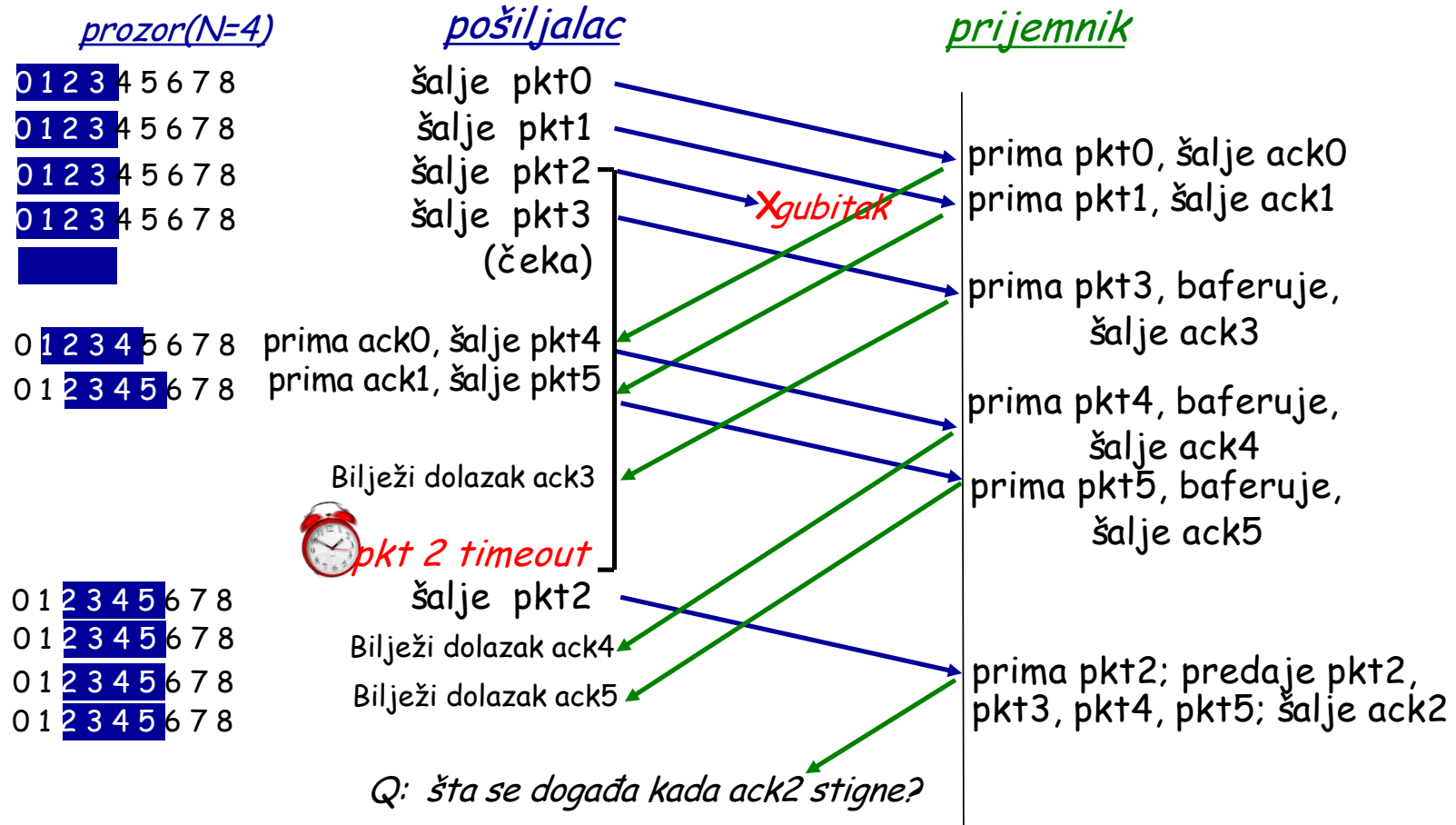
paket n u [rcvbase-N, rcvbase-1]

- ACK(n)

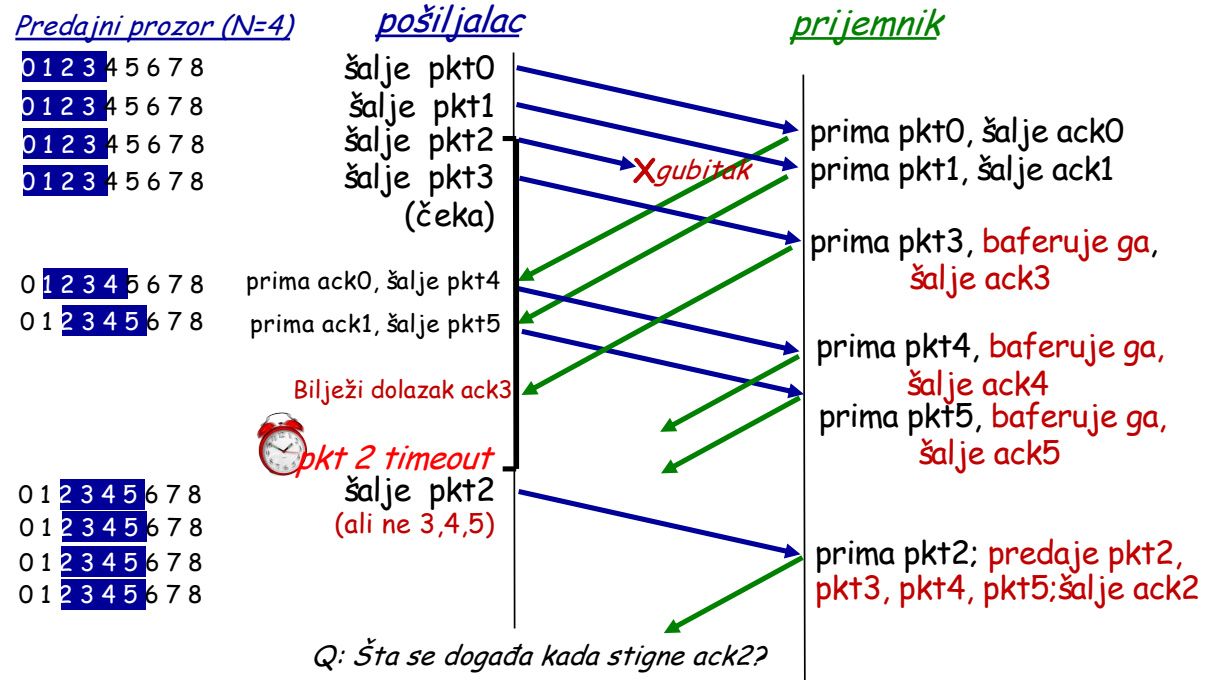
drugačije:

- ignoriši

Selective repeat u akciji



Selective repeat u akciji

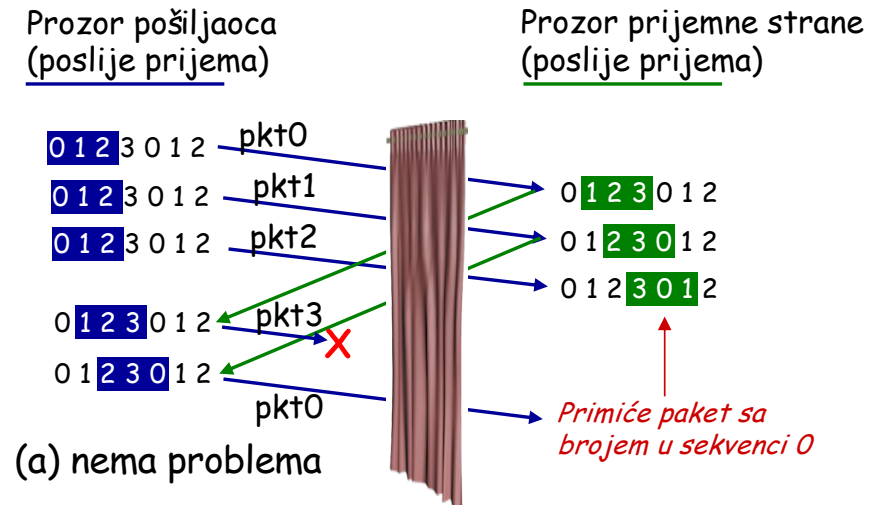


Selective repeat: dilema

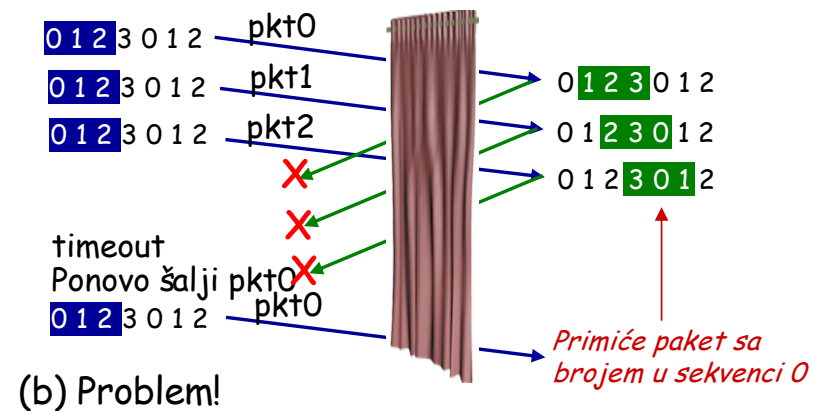
primjer:

- ❑ Brojevi u sekvenci: 0, 1, 2, 3
- ❑ Veličina prozora=3
- ❑ Prijemnik ne vidi razliku u dva scenarija!
- ❑ Duplikat se prima kao novi (b)

Kakva je relacija između veličine broja u sekvenci i veličine prozora kako bi se izbjegao problem pod (b)?

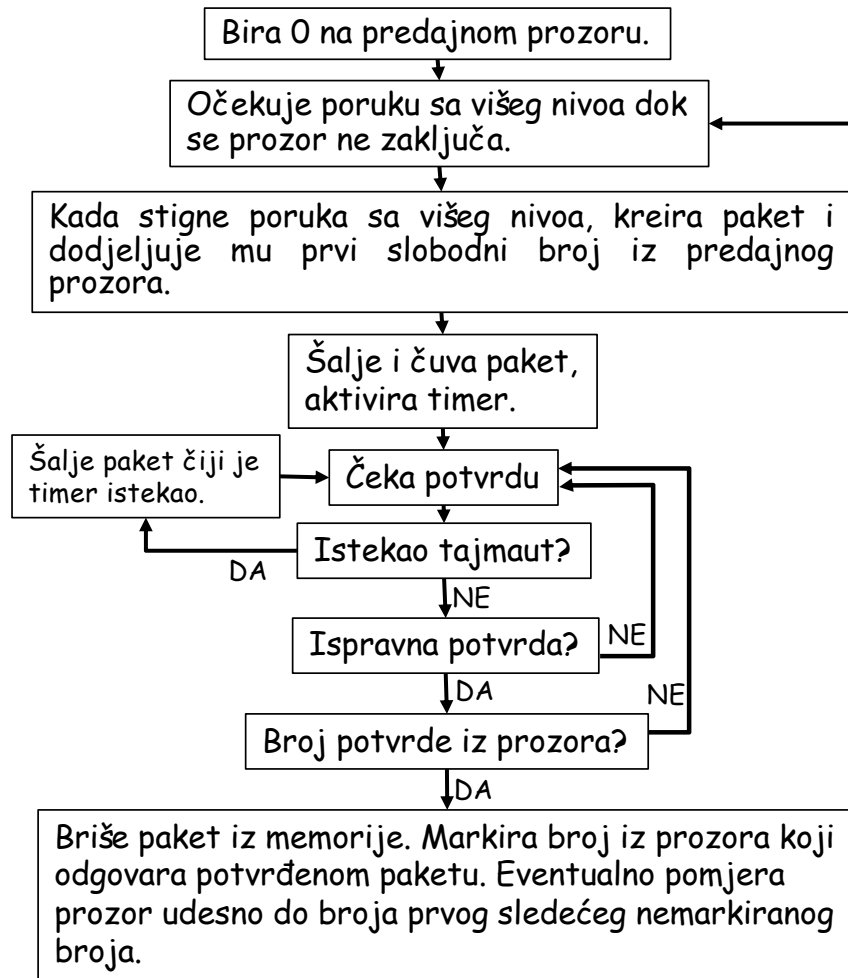


*Prijemnik ne vidi stranu pošiljaoca.
Prijemnikovo ponašanje je identično u oba slučaja!
Nešto nije u redu!*

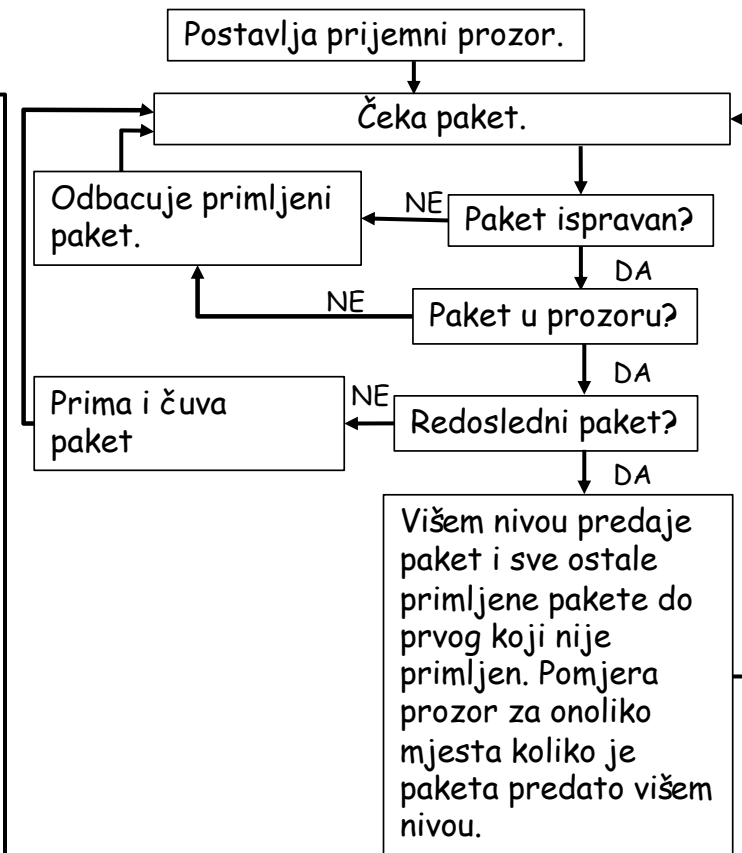


SR

Pošiljalac:



Prijemnik:



4. Nivo transporta

4.1 Servisi nivoa transporta

4.2 Multipleksiranje i demultipleksiranje

4.3 Nekonektivni transport: UDP

4.4 Principi pouzdanog prenosa podataka

4.5 Konektivni transport: TCP

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

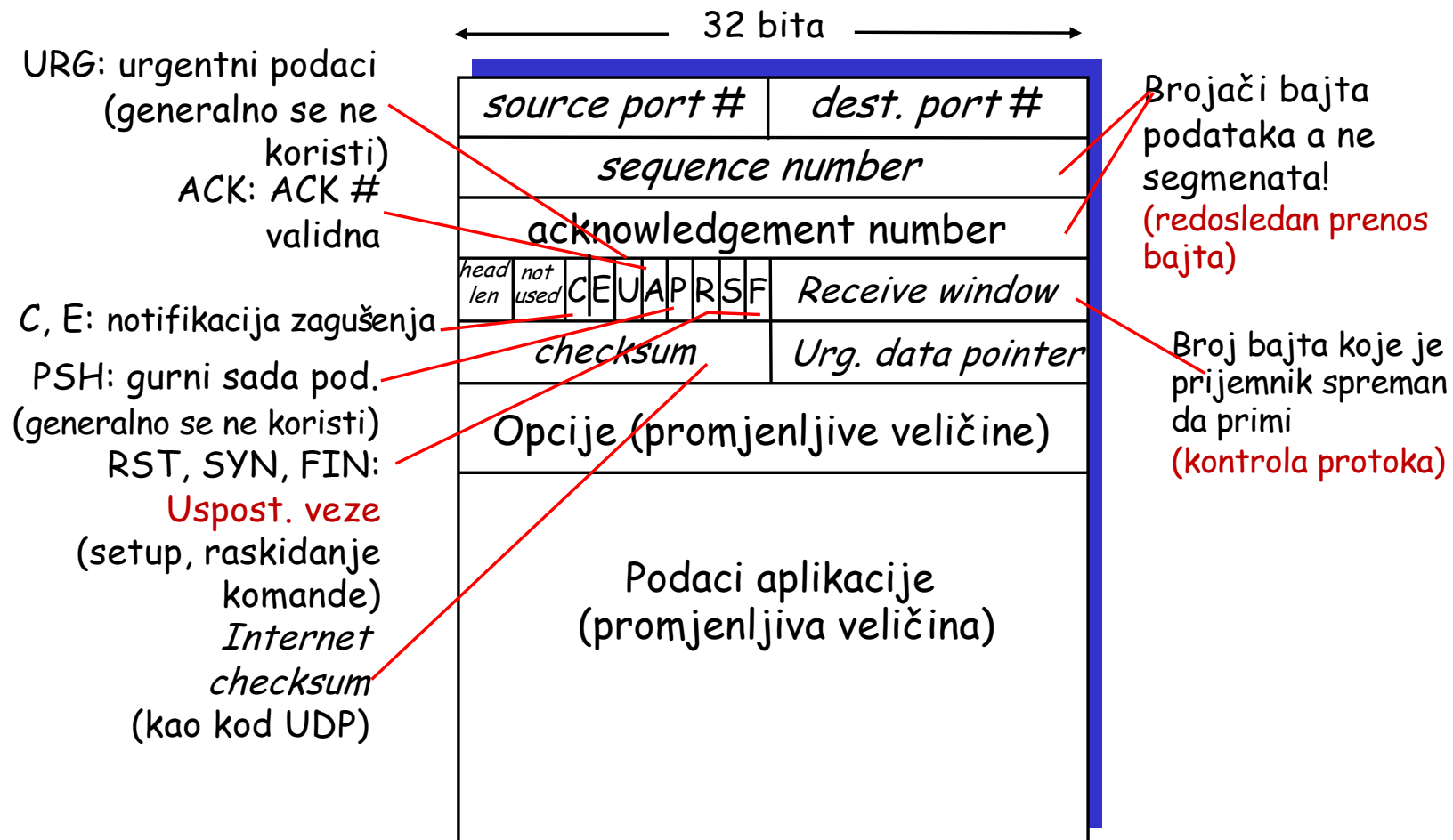
TCP: Pregled

RFC-ovi: 793, 1122, 1323, 2018,..., 7323

- ❑ tačka-tačka:
 - Jedan pošilj, jedan prij.
- ❑ pouzdan, redosledan prenos bajta:
 - nema "granica poruka"
- ❑ "pipelined":
 - TCP kontrola zagušenja i protoka podešava veličinu prozora
- ❑ *Baferi za slanje & prijem*
- ❑ Kumulativne potvrde
- ❑ "full duplex" prenos:
 - U istoj vezi prenos se obavlja u oba smjera
 - MSS: maksimalna veličina polja podataka u segmentu
- ❑ konektivan:
 - "handshaking" (razmjena kontrolnih poruka) inicira je pošiljalac, razmjenjuje stanja prije slanja
- ❑ 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.

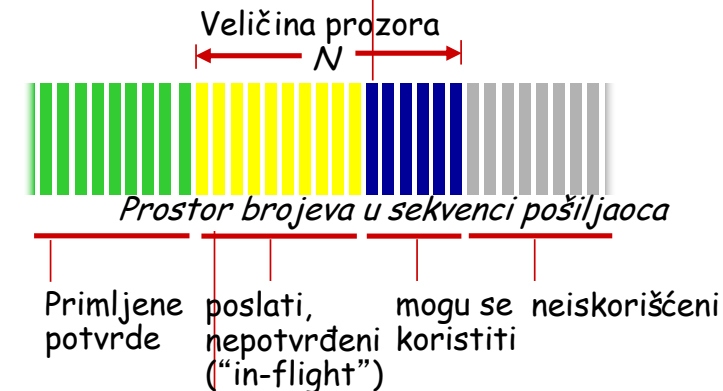
Potvrde (ACK):

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

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

Odlazni segment pošiljaoca

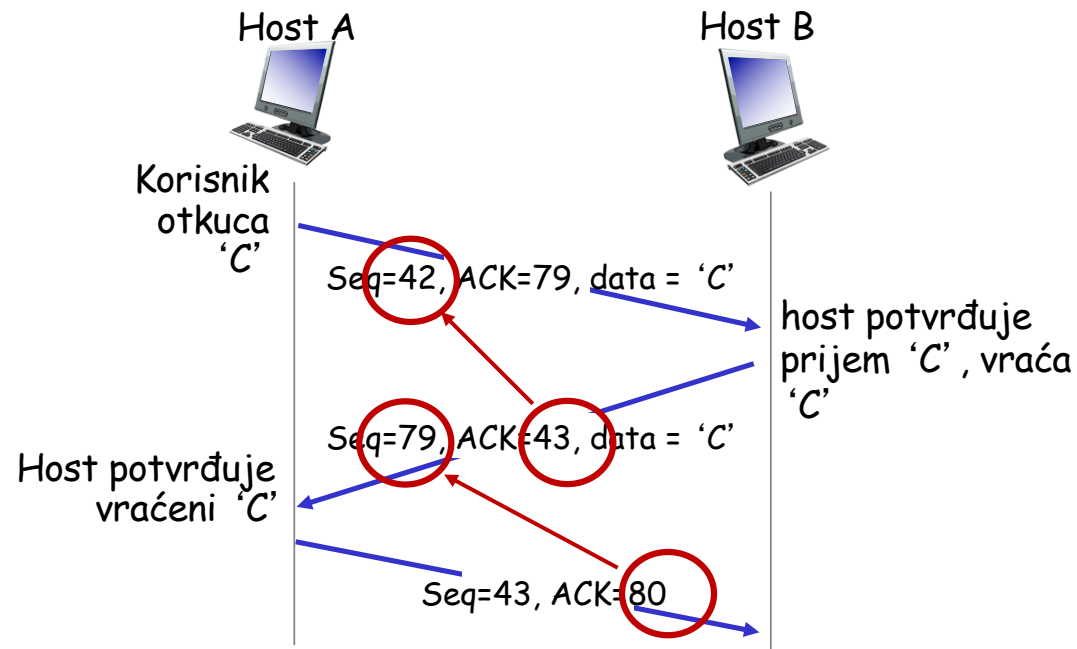
Source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer



Dolazni segment pošiljaoca

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

TCP brojevi u sekvenci, potvrde



jednostavan scenario (TELNET)

3. Nivo transporta

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

3.4 Principi pouzdanog prenosa podataka

3.5 Konektivni transport: TCP

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

TCP pouzdani prenos podataka

- TCP kreira pouzdani servis korišćenjem IP nepouzdanog servisa
- *Pipelined* segmenti
- Kumulativne potvrde
- TCP koristi jedan retransmisioni tajmer
- Retransmisije su triggerovane sa:
 - *timeout* događajima
 - duplim ack-ovima
- Na početku treba razmotriti pojednostavljenog TCP pošiljaoca:
 - Ignorišu se duplirani ack-ovi
 - Ignorišu se kontrole protoka i zagušenja

Događaji vezani za TCP pošiljaoca

1. Podaci primljeni od aplikacije:

- ❑ Kreiranje segmenta sa sekvencom brojeva
- ❑ Broj u sekvenci je redni 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
 - Napraviti odgovarajući *update*
 - Startovati tajmer ako postoje segmenti koji čekaju

TCP Round Trip Time i Timeout

Kako postaviti TCP vrijeme *timeout-a*?

- ❑ Duže od RTT-a
 - ali RTT varira
- ❑ Suviše kratko: prerani *timeout*
 - nepotrebne retransmisije
- ❑ Previše dugo: spora reakcija na gubitak segmenta
- ❑ Potrebna je aproksimacija RTT-a

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**

Da li **SampleRTT** vezivati za svaki nepotvrđeni segment?

Zašto ignorirati retransmisije?

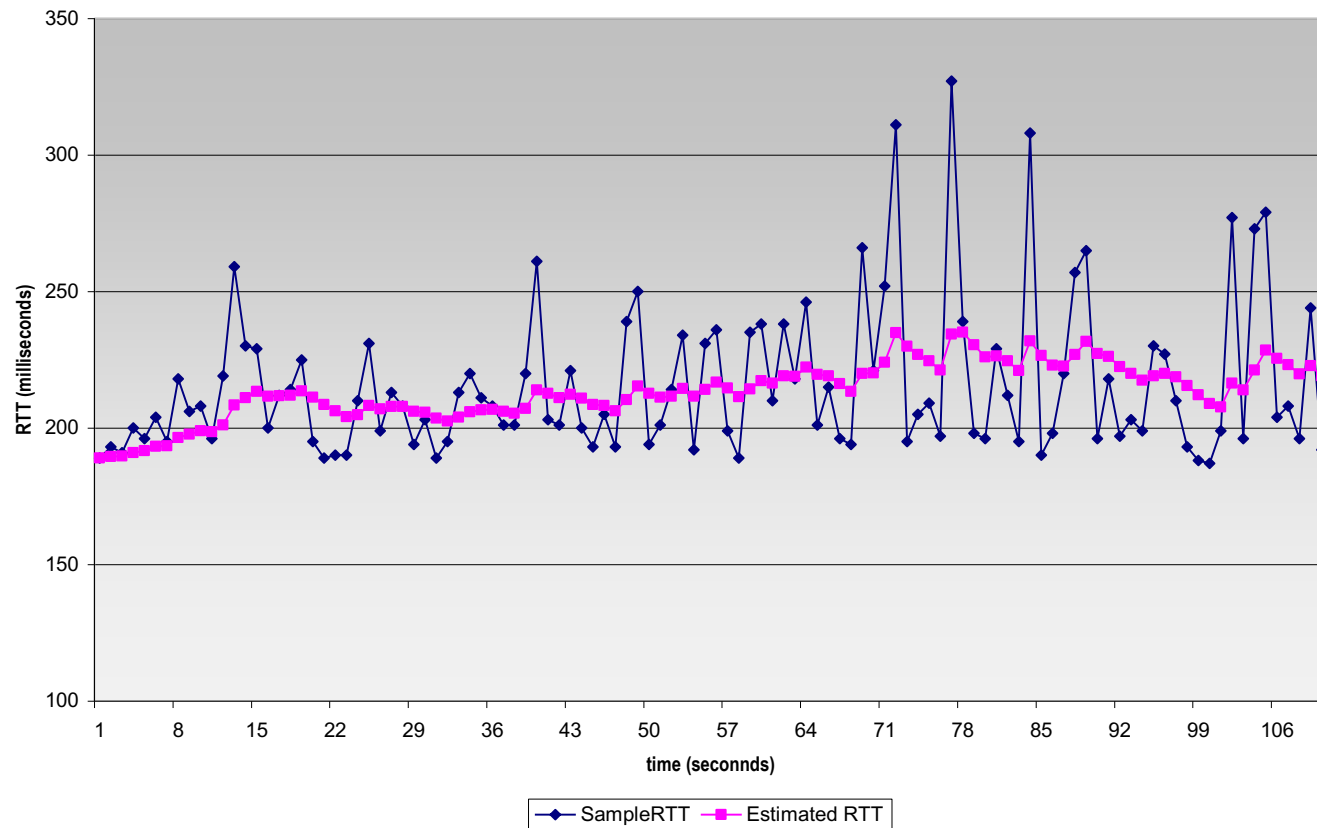
TCP Round Trip Time and Timeout

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

- 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

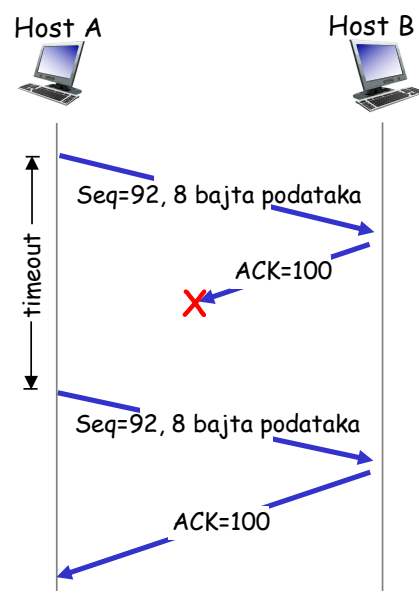
Tada se setuje timeout interval:

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

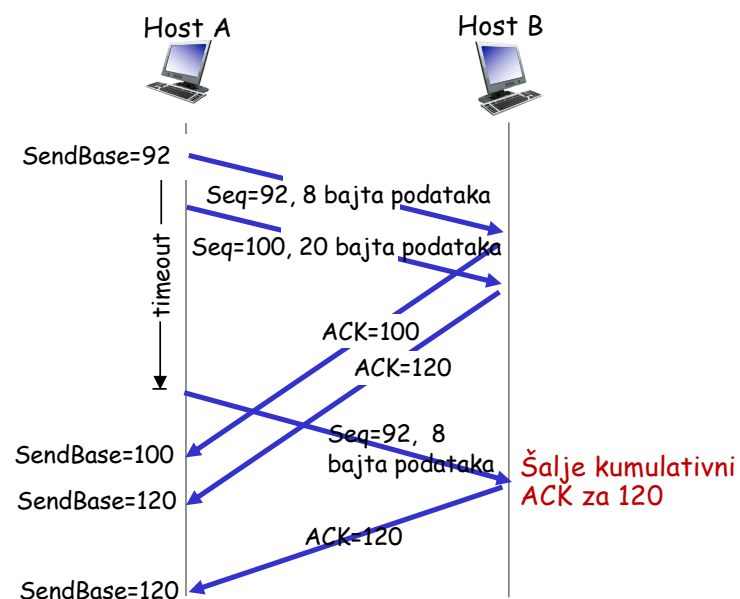
TCP generisanje ACK [RFC 1122, RFC 2581]

Događaj na prijemu	TCP akcije prijemnika
Dolazak <i>in-order</i> segmenta sa očekivanim brojem u sekvenci. Svi podaci do očekiv. broja su potvrđ.	ACK sa kašnjenjem. Čeka do 500ms za sledeći segment. Ako nema sledećeg, šalje ACK.
Dolazak <i>in-order</i> segmenta sa očekiv. brojem u sekvenci. Potvrđ. prijema drugog segmenta u toku.	Odmah šalje jednu kumulativnu ACK, potvrđujući oba <i>in-order</i> segmenta
Dolazak <i>out-of-order</i> segmenta sa većom vrijednosti broja u sekv. od očekivane. Detektovan prekid.	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

Scenariji retransmisije kod TCP-a

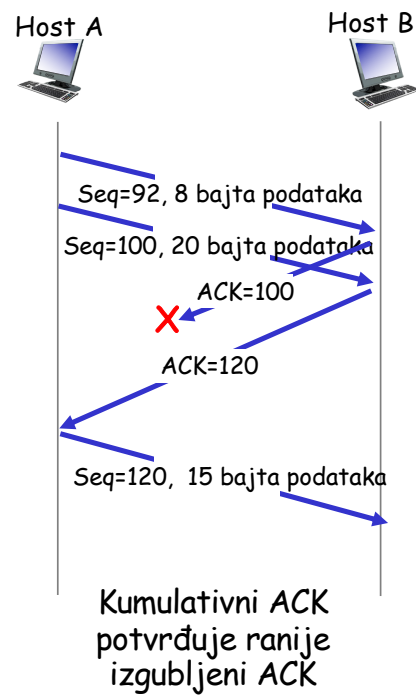


Scenario gubitka
ACK



Nedovršeni timeout (nema
retransmisije drugog dok ne
stigne potvrda za prvi)

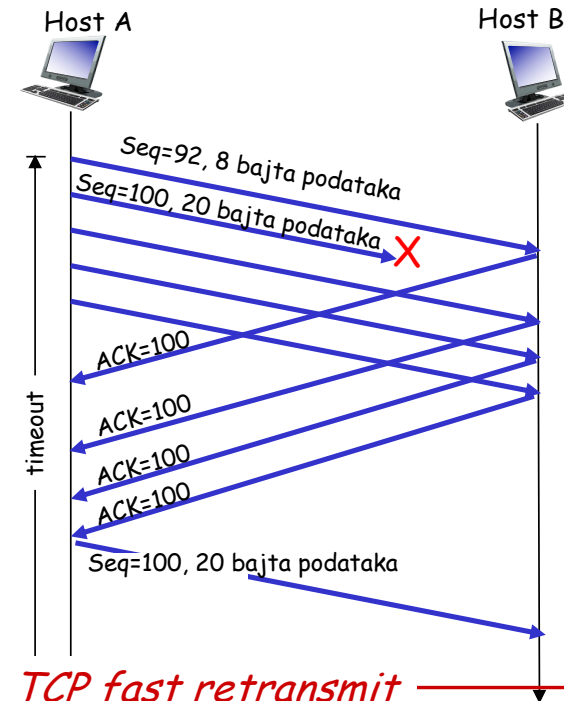
Scenariji retransmisije kod TCP-a



"Fast Retransmit"

- *Timeout* period je često predugačak:
 - Dugo kašnjenje prije slanja izgubljenog paketa
- Detekcija izgubljenog segmenta preko dupliranih ACK-ova.
 - Pošiljalac često šalje mnogo segmenata
 - Ako je segment izgubljen, najvjerojatnije će biti dosta duplih ACK-ova.

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



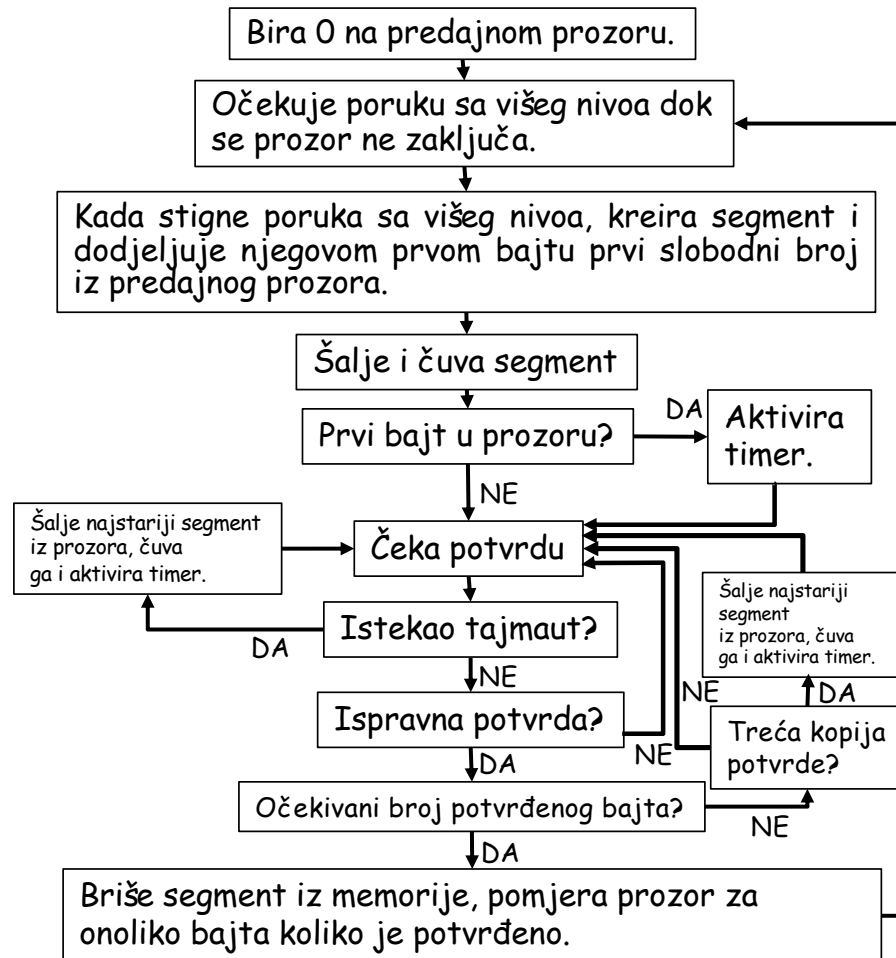
TCP fast retransmit

Ako pošiljalac primi 3 ACK za isti segment, pretpostavlja se da je segment poslije potvrđenog izgubljen:

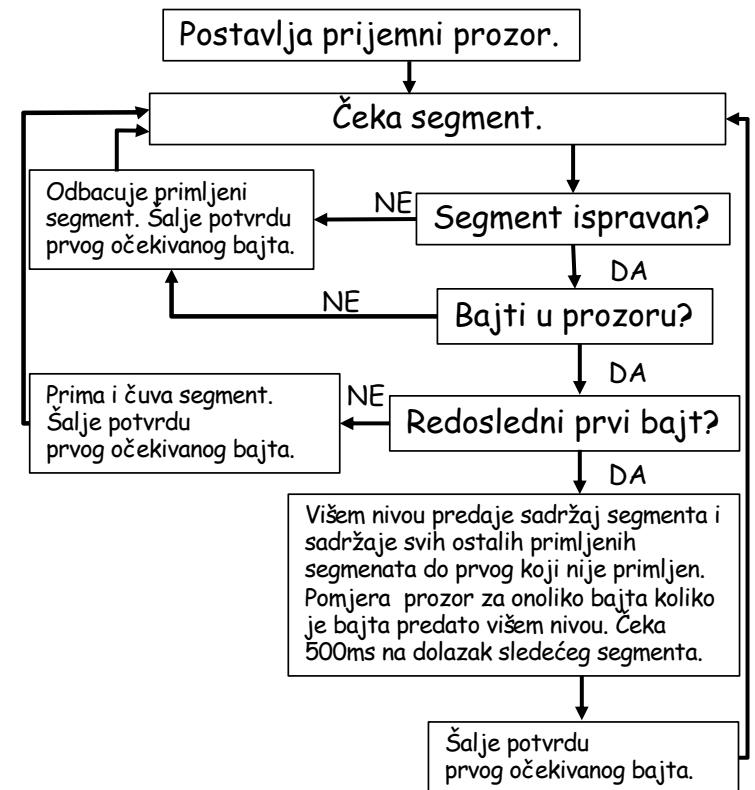
fast retransmit: ponovljeno slanje segmenta prije nego što je tajmer istekao

TCP

Pošiljalac:



Prijemnik:



3. Nivo transporta

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

3.3 Nekonektivni transport: UDP

3.4 Principi pouzdanog prenosa podataka

3.5 Konektivni transport: TCP

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

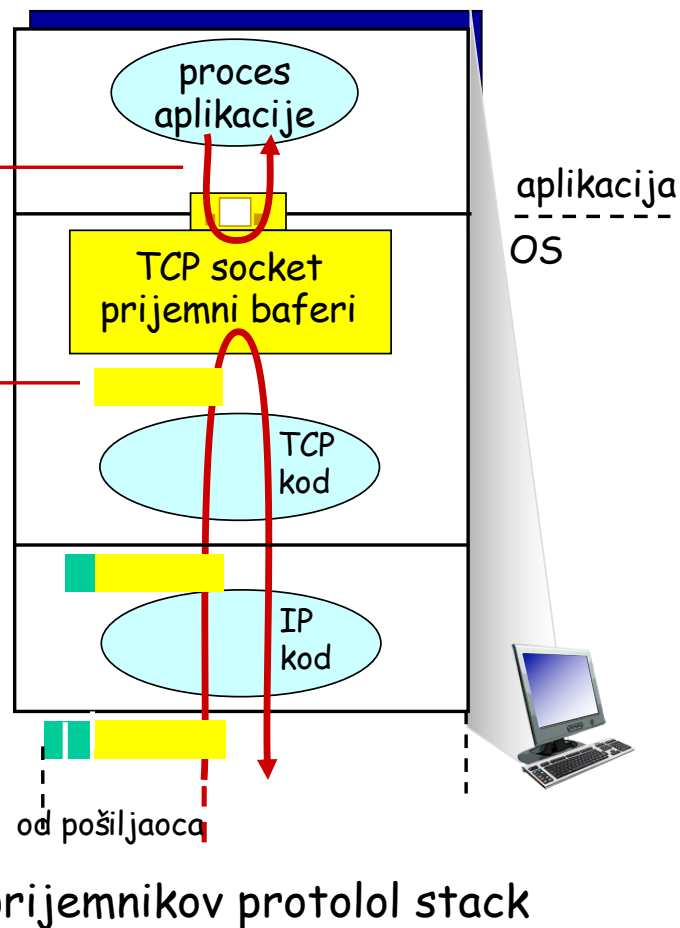
TCP kontrola protoka

Kontrola protoka

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

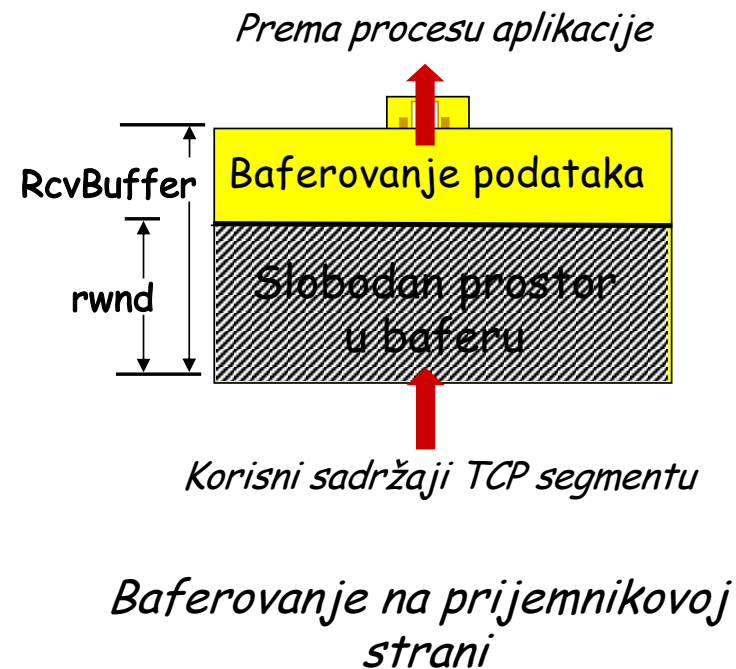
Aplikacija može ukloniti podatke iz bafera TCP socketa

... sporije od predaje podataka koje prima TCP (pošiljalac šalje)



TCP kontrola protoka

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



3. Nivo transporta

3.1 Servisi nivoa transporta

3.2 Multipleksiranje i demultipleksiranje

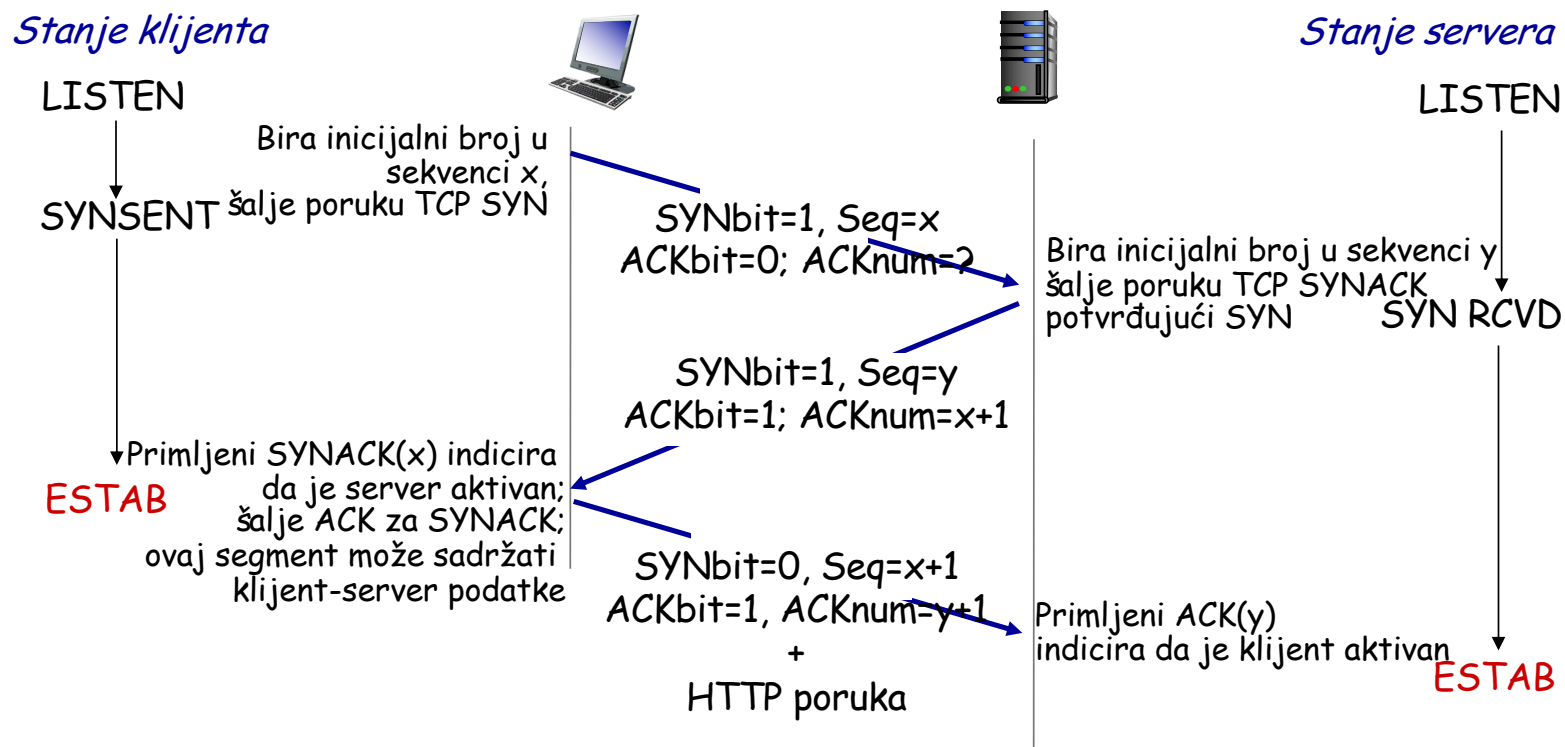
3.3 Nekonektivni transport: UDP

3.4 Principi pouzdanog prenosa podataka

3.5 Konektivni transport: TCP

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

Uspostavljanje TCP konekcije (3-way handshake)



Zatvaranje TCP 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=y

FINbit=0, seq=y
ACKbit=1; ACKnum=x+1

FINbit=1, seq=z
ACKbit=1; ACKnum=w

FINbit=0, seq=w
ACKbit=1; ACKnum=y+1

Još uvijek
može slati

Ne može
više slati

Stanje servera

ESTAB

CLOSE_WAIT

LAST_ACK

CLOSED