UNIVERZITET CRNE GORE

RAČUNARSKE MREŽE

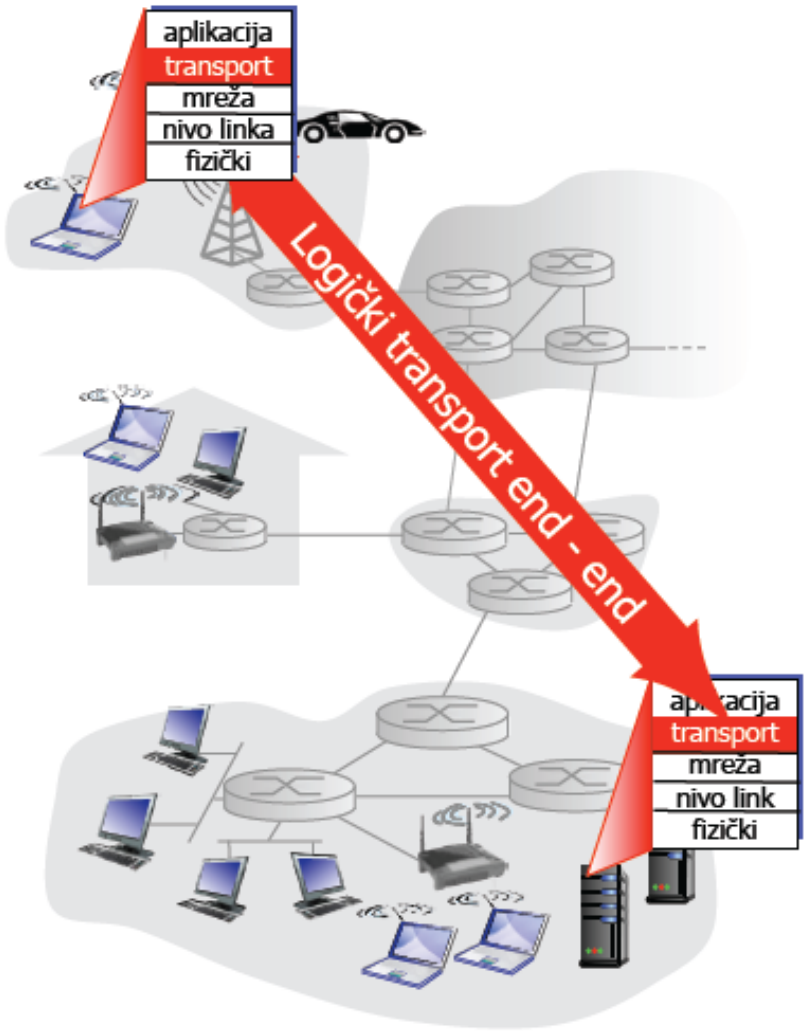
Doc. dr Uglješa Urošević

ugljesa@ucg.ac.me

Nivo transporta

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
  - Internet: TCP i UDP



Poređenje transportnog i mrežnog nivoa

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

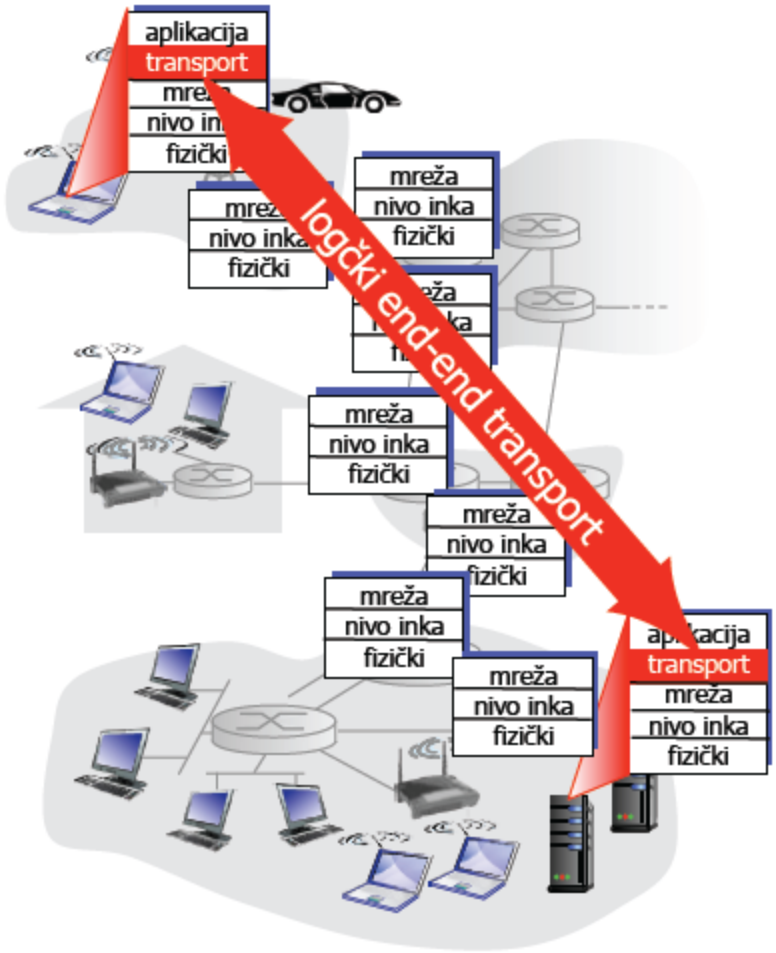
Analogija:

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

Internet protokoli transportnog nivoa

- pouzdana, redosledna isporuka (TCP)
  - Kontrola zagušenja
  - Kontrola protoka
  - Uspostavljanje veze
- nepouzdana, neredosledna isporuka: UDP
  - Bez unapređenja"best-effort" pristupa IP
- Servisi koji se ne pružaju:
  - Garantovano kašnjenje
  - Garantovana propusnost



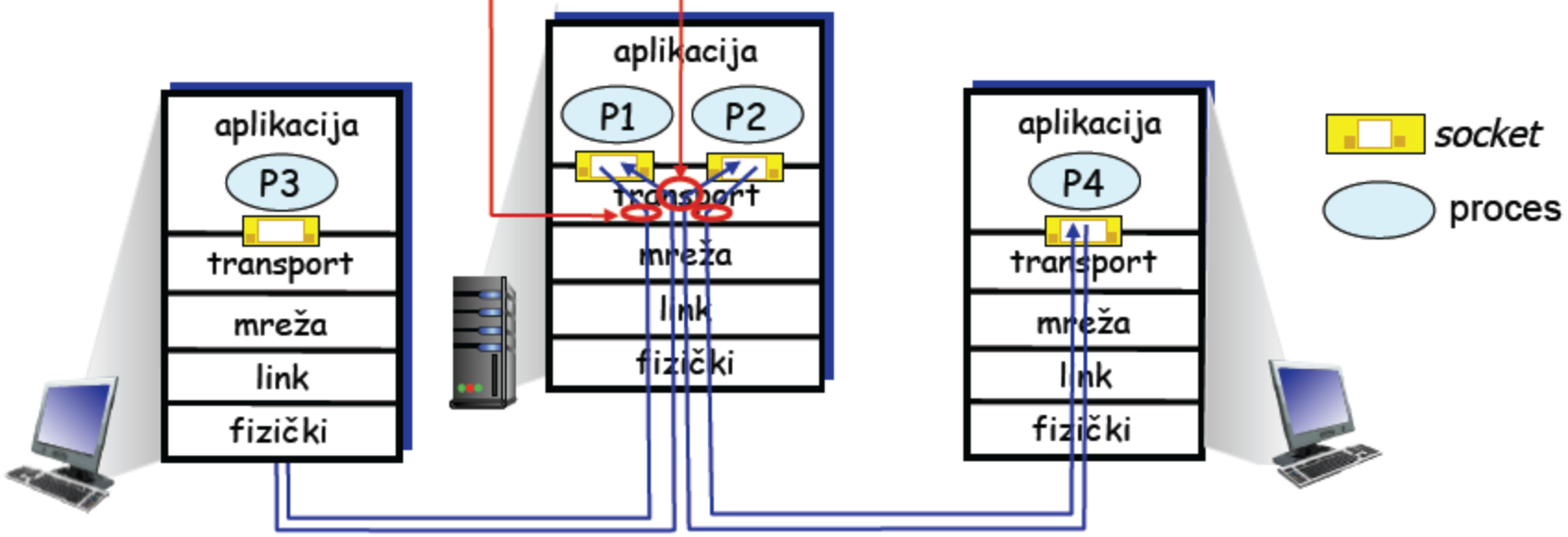
Multipleksiranje/demultipleksiranje

Multipleksiranje na predaji:

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

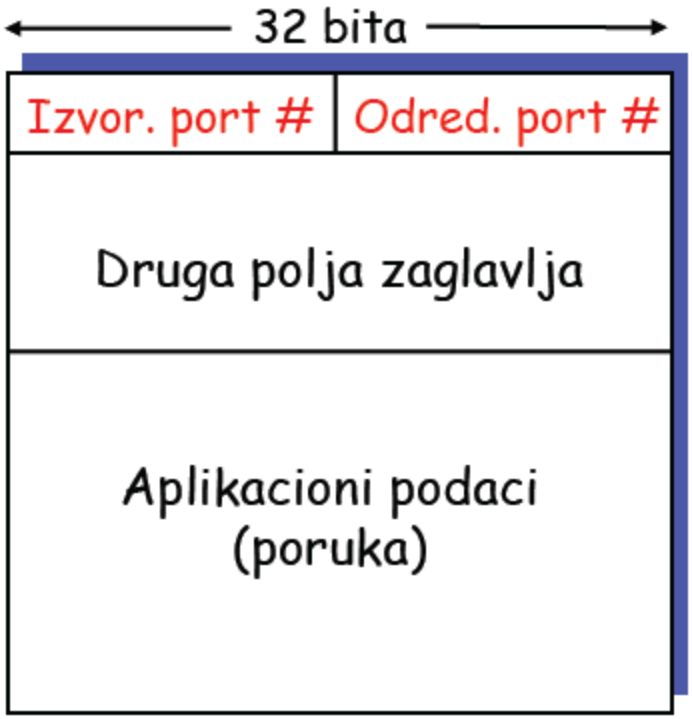
Demultipleksiranje na prijemu:

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)
- host koristi IP adrese & brojeve portova da usmjeri segment na odgovarajući socket



TCP/UDP format segmenta

Dobro poznati portovi

PROTOCOL	PORT
FTP data port (Active Mode)	TCP 20
FTP control port	TCP 21
SSH	TCP 22
SCP (uses SSH)	TCP 22
SFTP (uses SSH)	TCP 22
Telnet	TCP 23
SMTP	TCP 25
TACACS+	TCP 49
DNS name queries	UDP 53
DNS zone transfers	TCP 53
TFTP	UDP 69
HTTP	TCP 80
Kerberos	UDP/TCP 88
POP3	TCP 110
SNMP	UDP 161

Dobro poznati portovi

SNMP trap	UDP 162
NetBIOS (TCP rarely used)	TCP/UDP 137, 138, and TCP 139
IMAP4	TCP 143
LDAP	TCP 389
HTTPS	TCP 443
SMTP with SSL/TLS	TCP 465
IPsec (for VPN with IKE)	UDP 500
LDAPv2 using SSL	TCP 636
LDAPv3 using TLS	TCP 636
IMAP using SSL/TLS	TCP 993
POP using SSL/TLS	TCP 995
L2TP	UDP 1701
PPTP	TCP 1723
RDP	TCP/UDP 3389
Microsoft SQL Server	TCP 1433

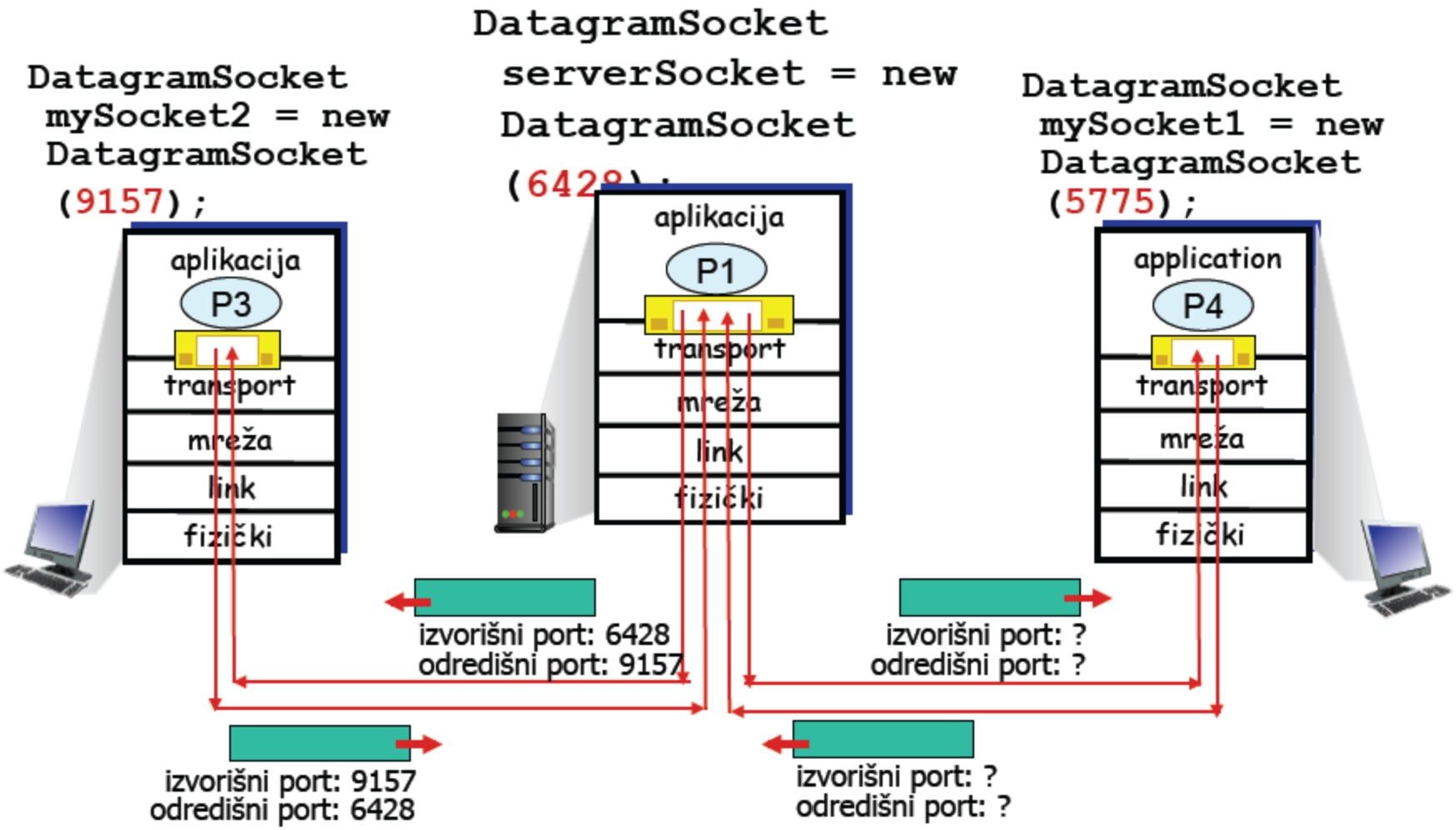
Nekonektivno demultipleksiranje (UDP)

- 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*

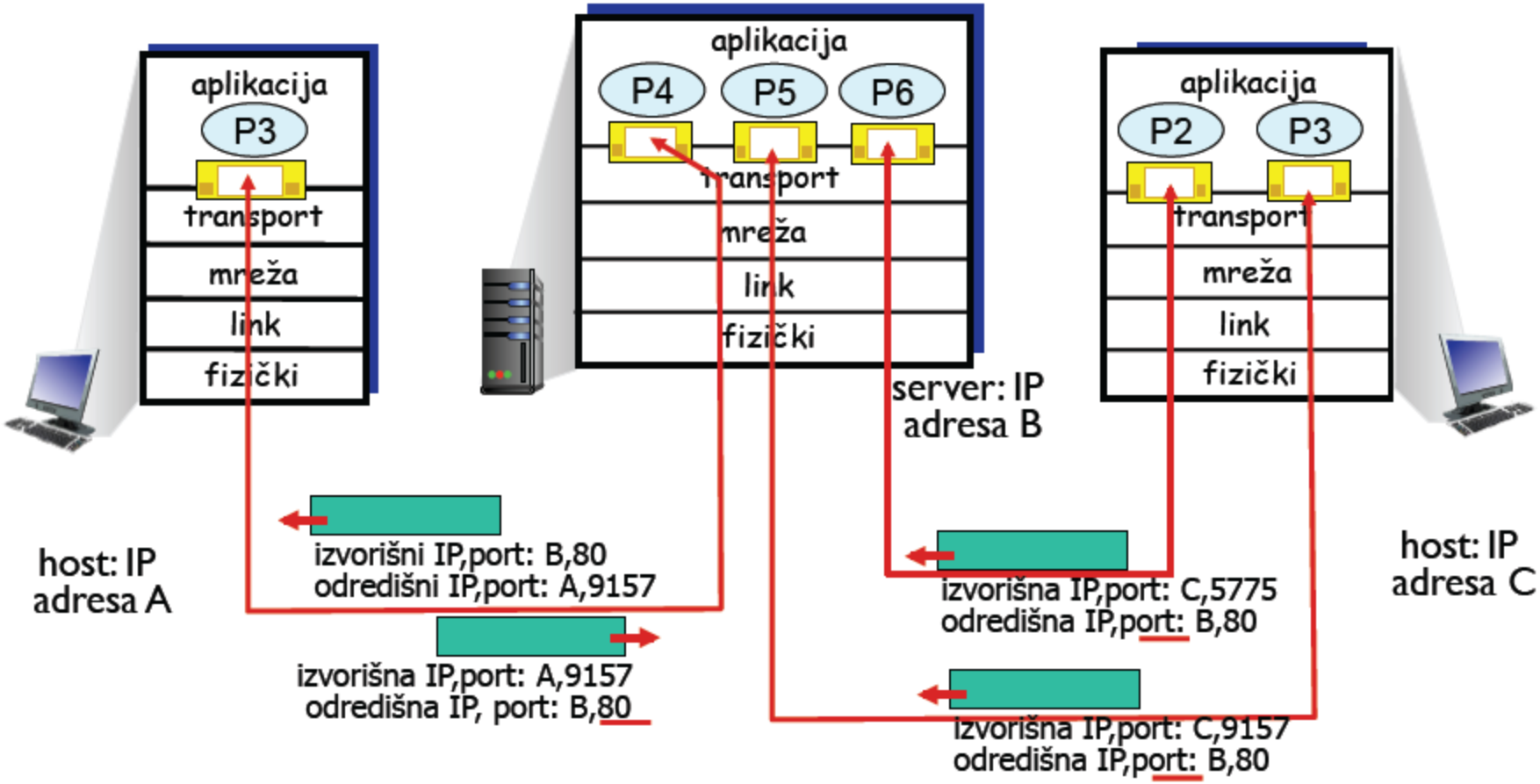
Nekonektivno multipleksiranje



Konektivno demultipleksiranje

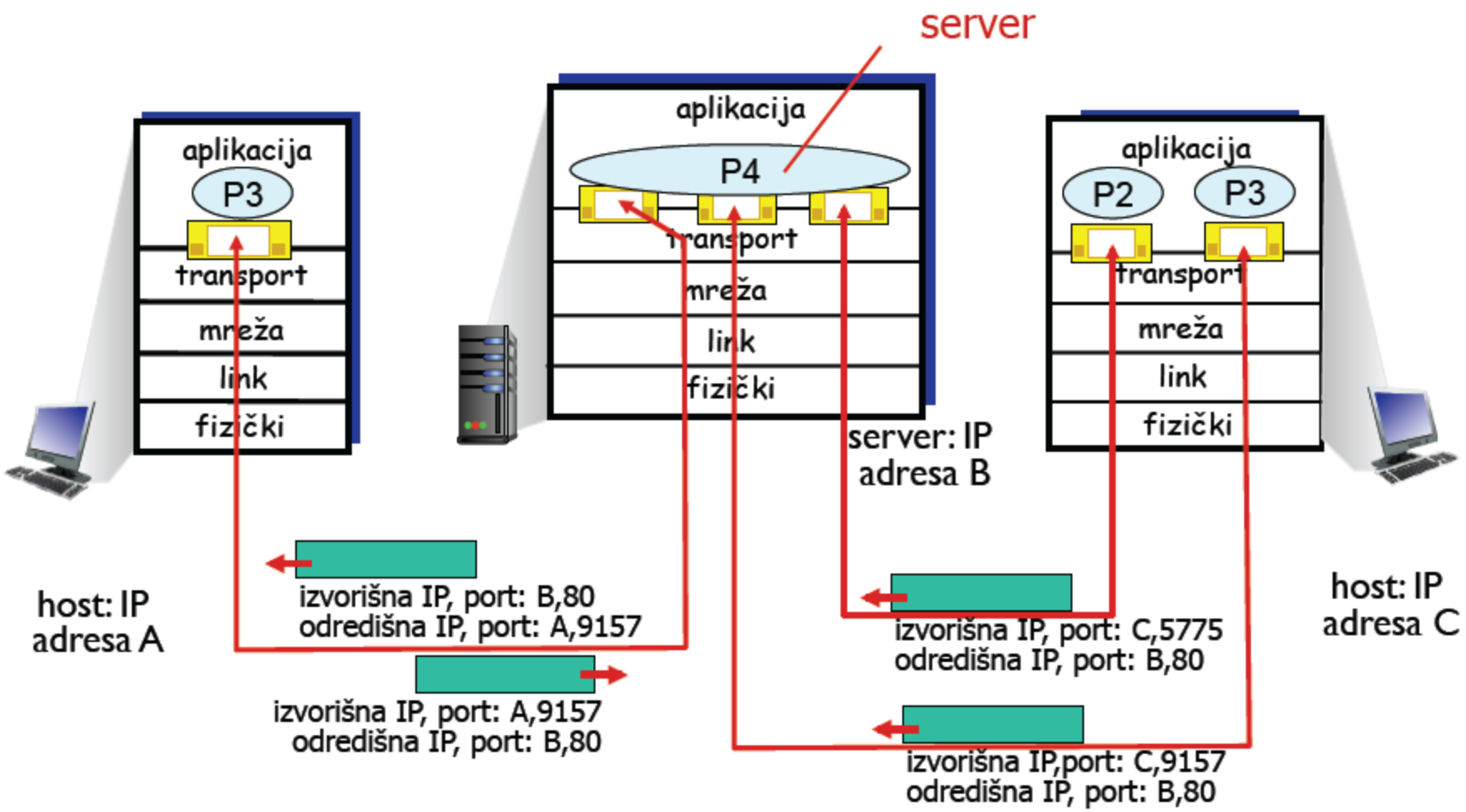
- TCP *socket* identifikuju 4 parametra:
  - Izvorišna IP adresa
  - Izvorišni broj porta
  - Odredišna IP adresa
  - Odredišni broj porta
- Prijemni host koristi sve četiri vrijednosti za usmjeravanje segmenta na odgovarajući socket
- Server host može podržavati više simultanih TCP *socket*-a:
  - svaki *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



Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Name translation	DNS	Typically UDP

UDP: User Datagram Protocol [RFC 768]

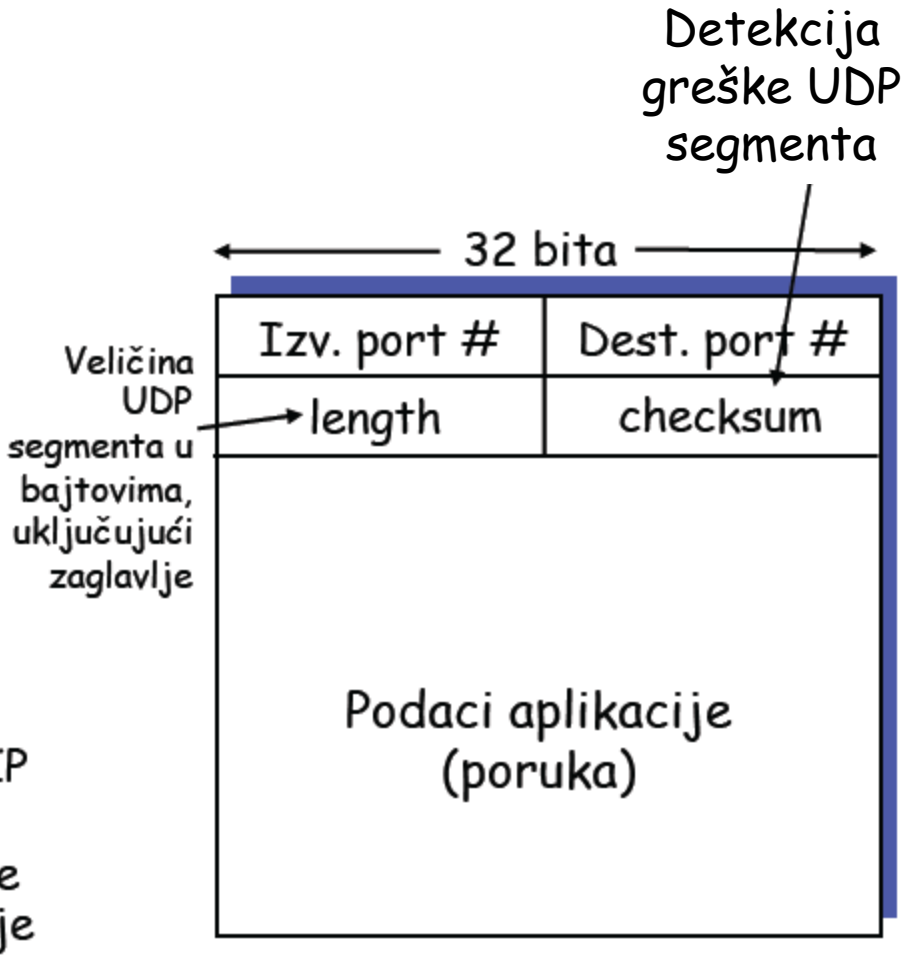
- Nema poboljšanja koja se nude Internet protokolu
- "*best effort*" servis, UDP segmenti mogu biti:
  - izgubljeni
  - neredosledno predati
- nekonektivni:
  - nema uspostavljanja veze (*handshaking*) između UDP pošiljaoca i prijemnika
  - svaki UDP segment se tretira odvojeno od drugih

Zašto onda UDP?

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

UDP: više

- 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)
  - RIP (zbog periodičnog slanja RIP update-a)
- Pouzdani prenos preko UDP: mora se dodati pouzdanost na nivou aplikacije
  - Oporavak od greške na nivou aplikacije
- Problem kontrole zagušenja je i dalje otvoren!



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. Šta više, greška može nastupiti i u nekom od rutera.

Pošiljac:

- Tretira sadržaj segmenta kao sekvence 16-bitnih prirodnih brojeva
- checksum: dodaje (suma 1. komplementa) informaciju segmentu
- Pošiljac postavlja vrijednost checksum -e u odgovarajuće polje UDP segmenta

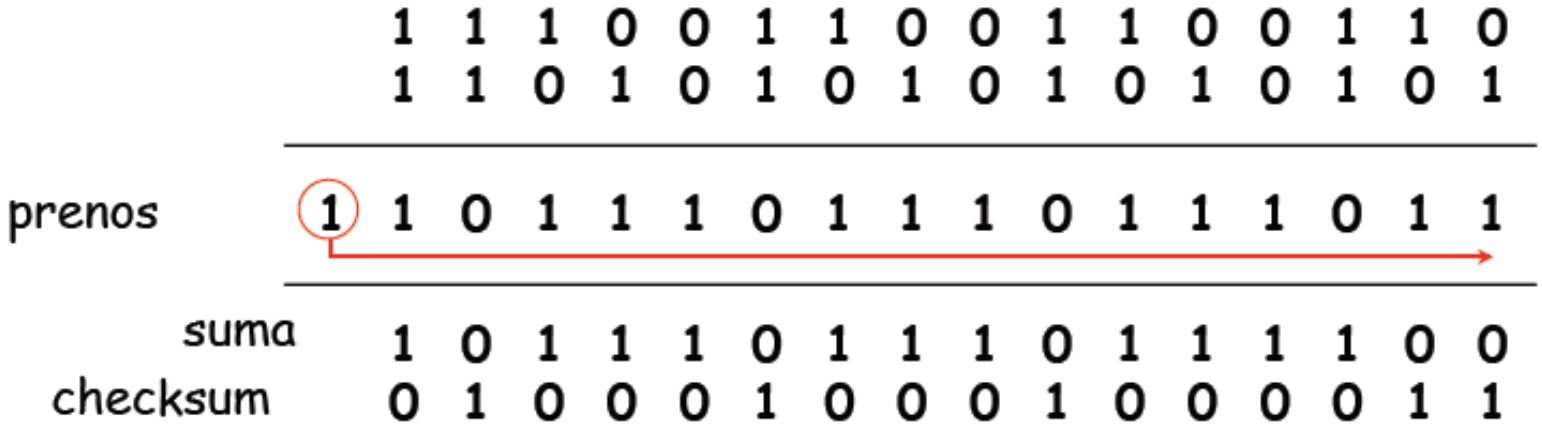
Prijemnik:

- Sumiraju se sekvence 16-bitnih brojeva (uključujući polje checksum) i posmatra se da li je rezultat broj koji sadrži 16 jedinica:
  - NE - detektovana greška
  - DA - nema greške. *Da li ste sigurni?*

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

Internet Checksum-a primjer

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



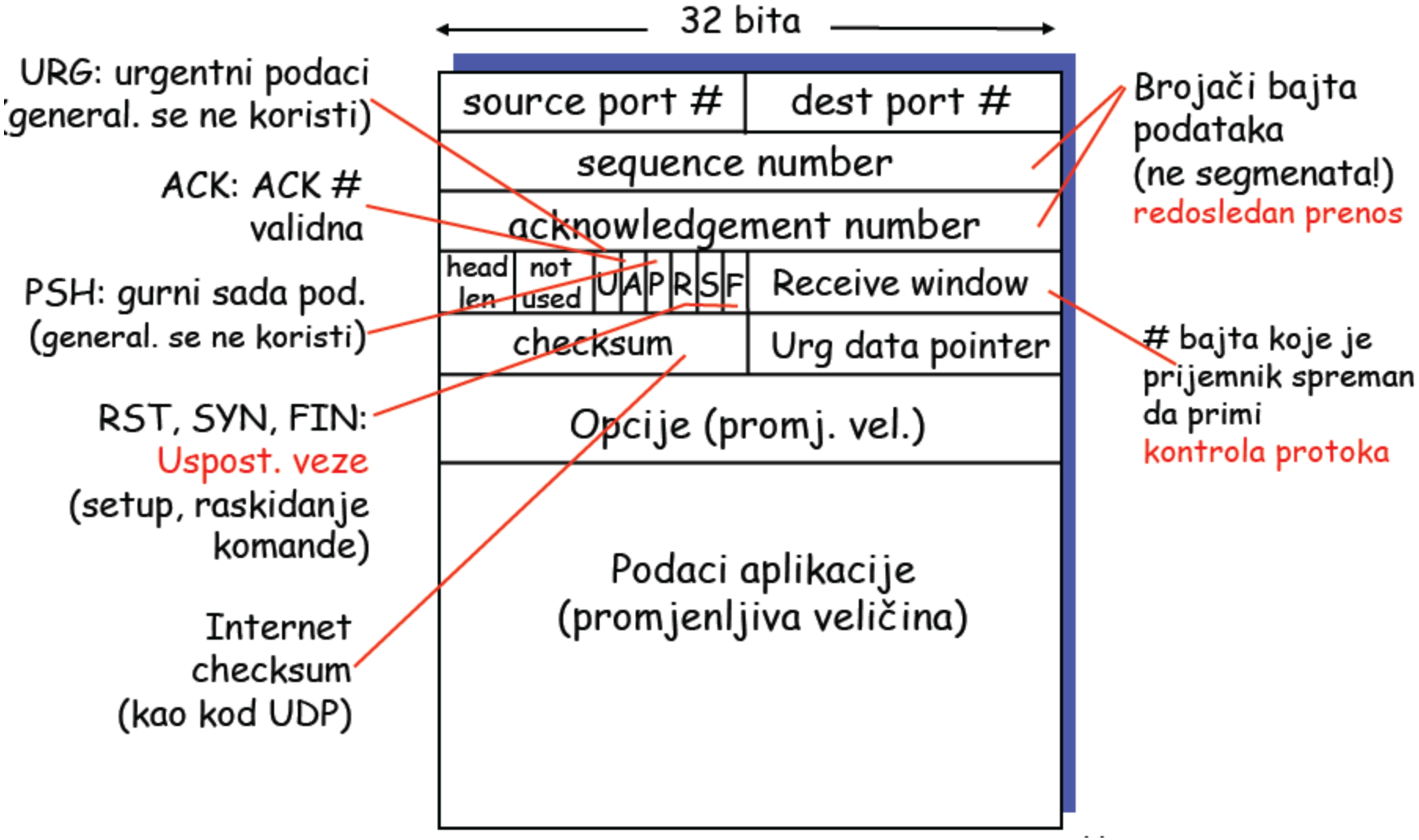
TCP: Pregled

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

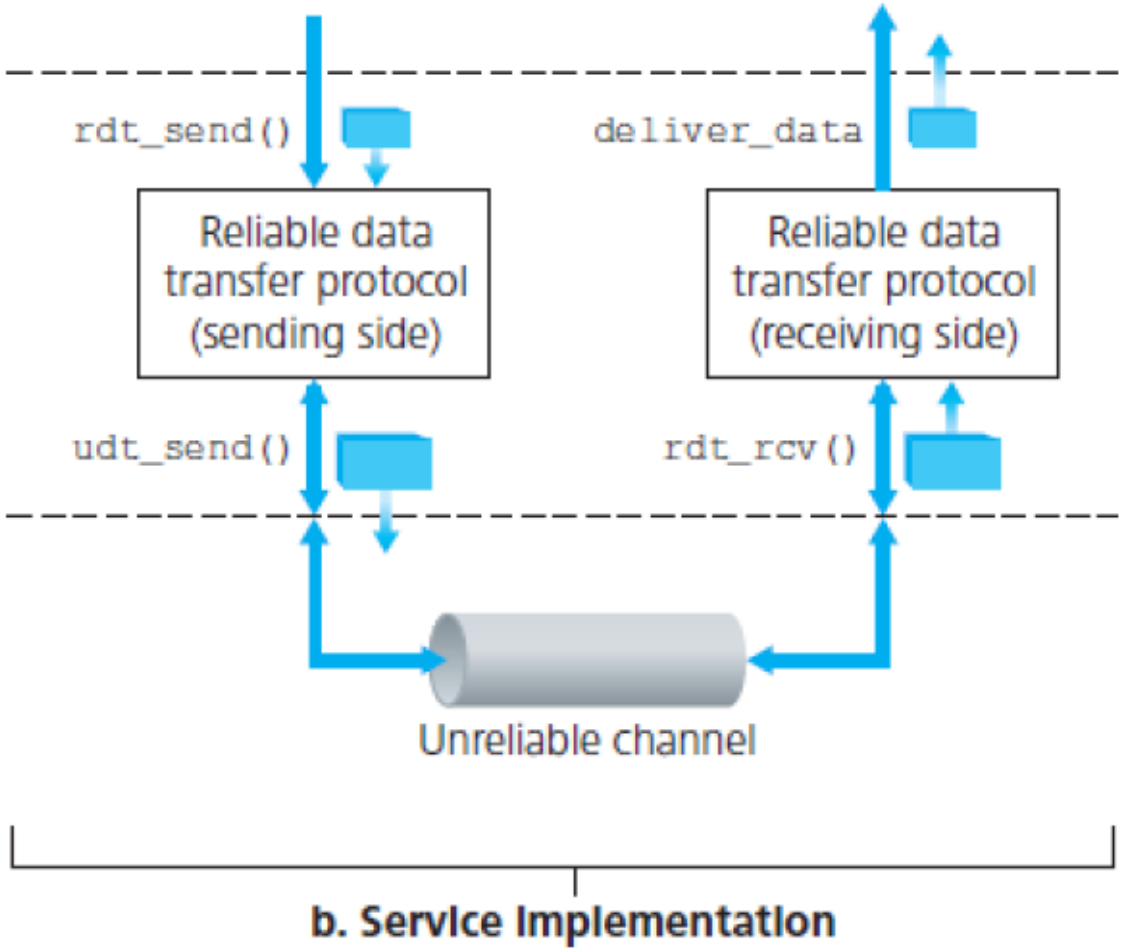
- tačka-tačka:
  - Jedan pošilj, jedan prij.
- pouzdan, redosledan prenos bajta:
  - nema "granica poruka"
- "pipelined":
  - TCP kontrola zagušenja i protoka podešava veličinu prozora
- *Baferi za slanje & prijem*

- "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 je pošiljalac, razmjenjuje stanja prije slanja
- kontrola protoka:
  - Pošiljalac ne može "zagušiti" prijemnika

TCP struktura segmenta (21B-1480B)



TCP pouzdani prenos podataka

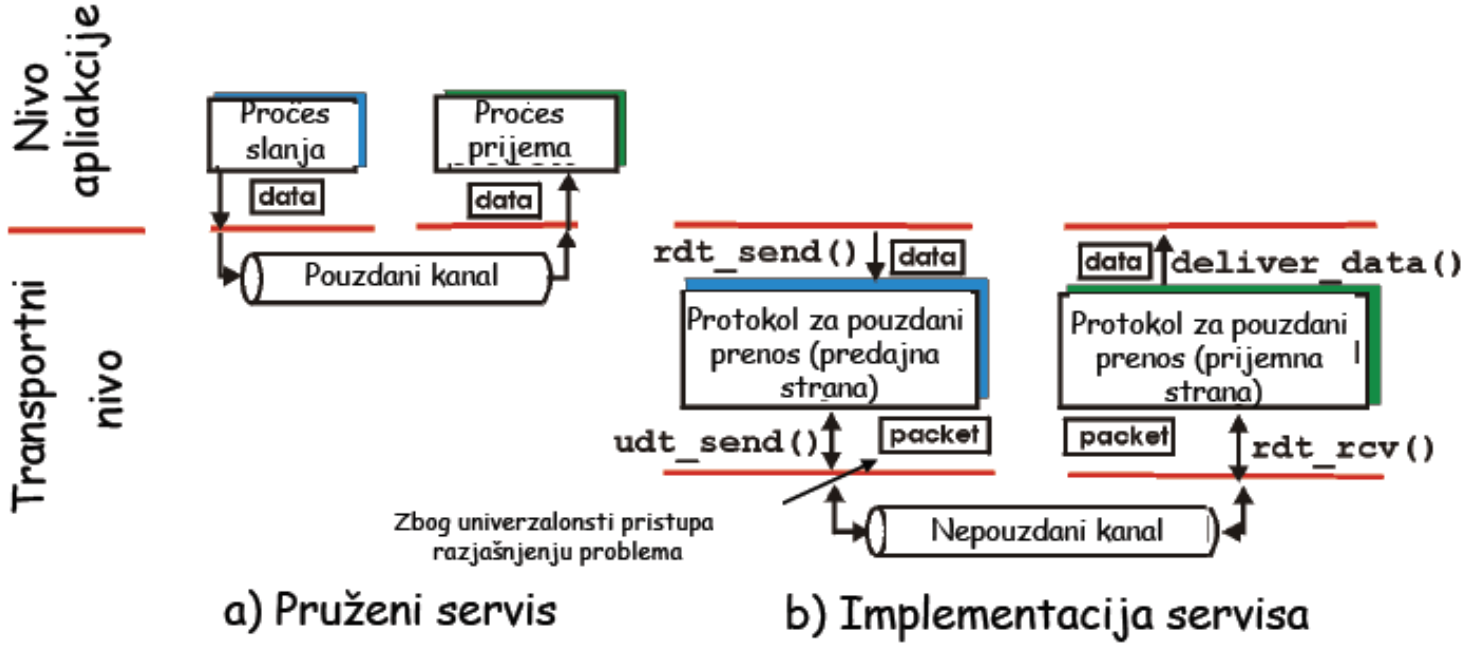


b. Service Implementation

rdt - reliable data transfer
udt - unreliable data transfer

Opšti principi pouzdanog prenosa podataka

- Važno na nivoima aplikacije, transporta, linka
- Jedna od top-10 karakteristika mreže!



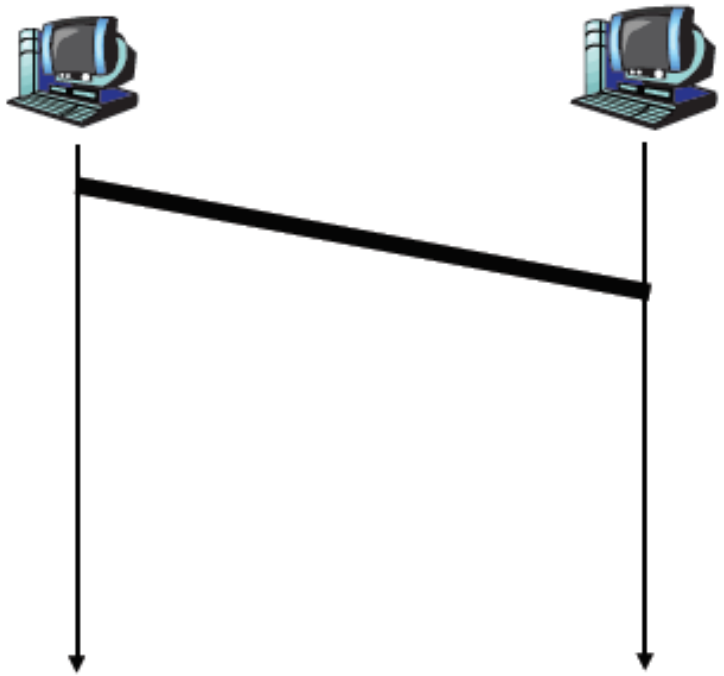
a) Pruženi servis b) Implementacija servisa

- Karakteristike nepouzdanog kanala će odrediti kompleksnost pouzdanog protokola za prenos podataka (rdt)

rdt - reliable data transfer
udt - unreliable data transfer

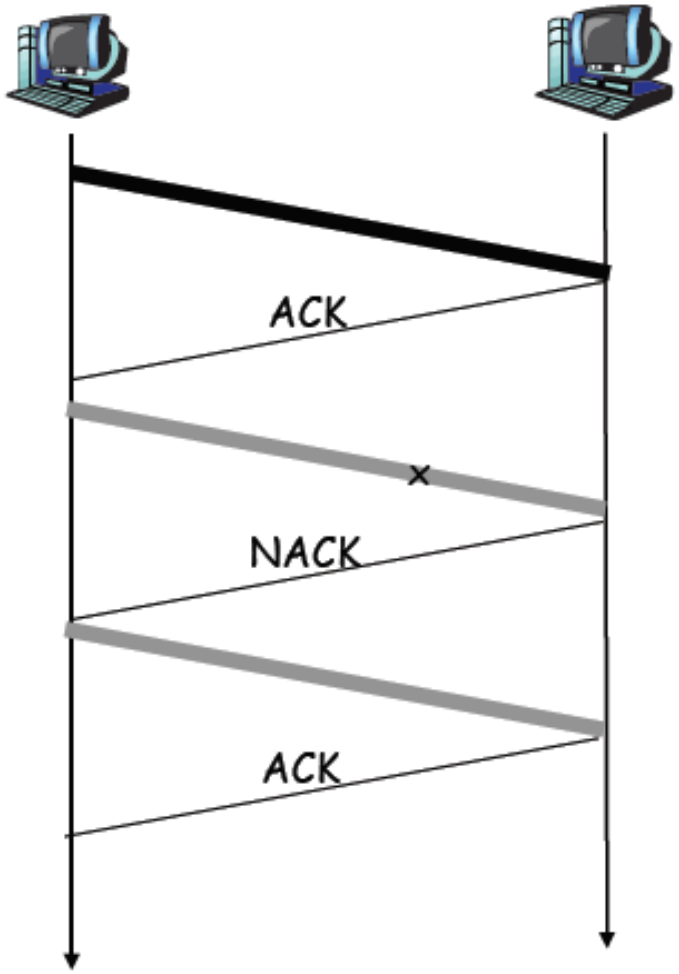
Pouzdani prenos preko pouzdanog kanala

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



Kanal sa greškom (ali nema gubitka paketa)

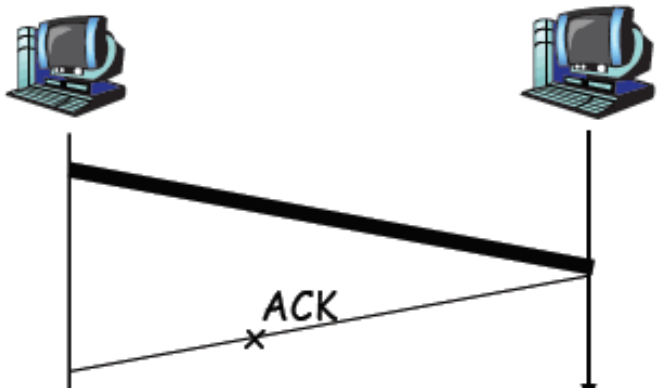
- Kanal može zamijeniti vrijednosti bita u paketu
- Potrebno je detektovati grešku na prijemnoj strani. Kako?
- Prijemna strana o tome mora obavijestiti predajnu stranu potvrdom uspješnog (ACK) ili neuspješnog prijema (NACK)
- Kada prijemna strana primi ACK šalje nove podatke, ako primi NACK obavlja ponovno slanje prethodnog paketa (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



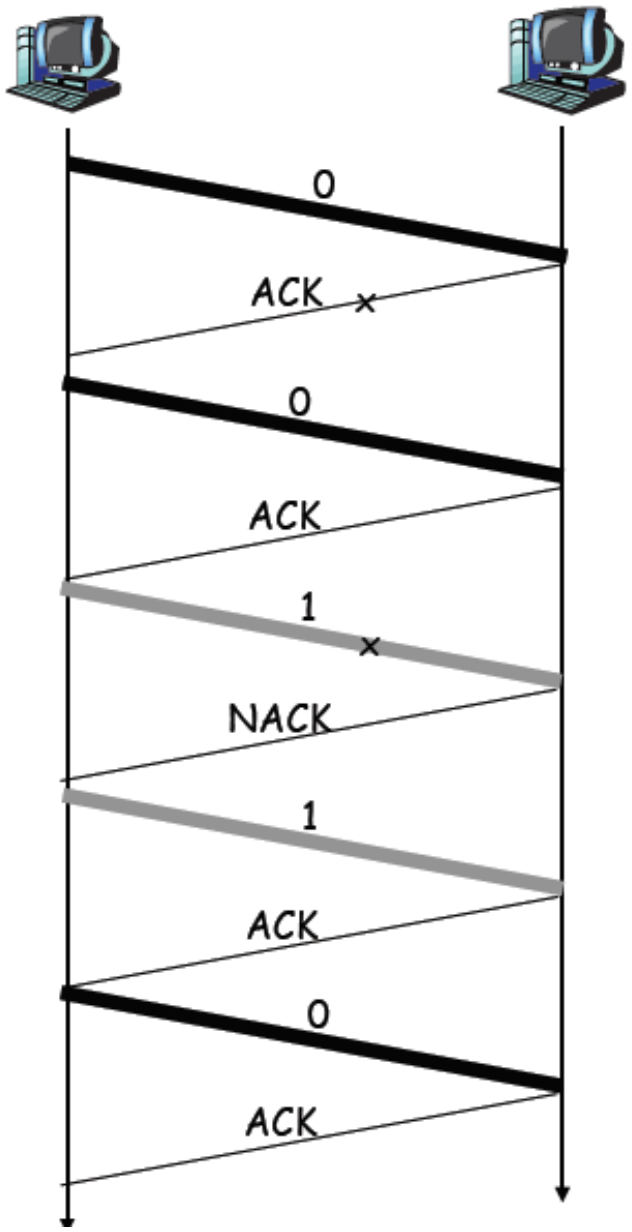
Rješavanje duplikata:

- 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 poslati paket.

STOP & WAIT

Pošiljac šalje jedan paket, a zatim čeka na odgovor

Stop & wait (u kanalu bez gubitaka)



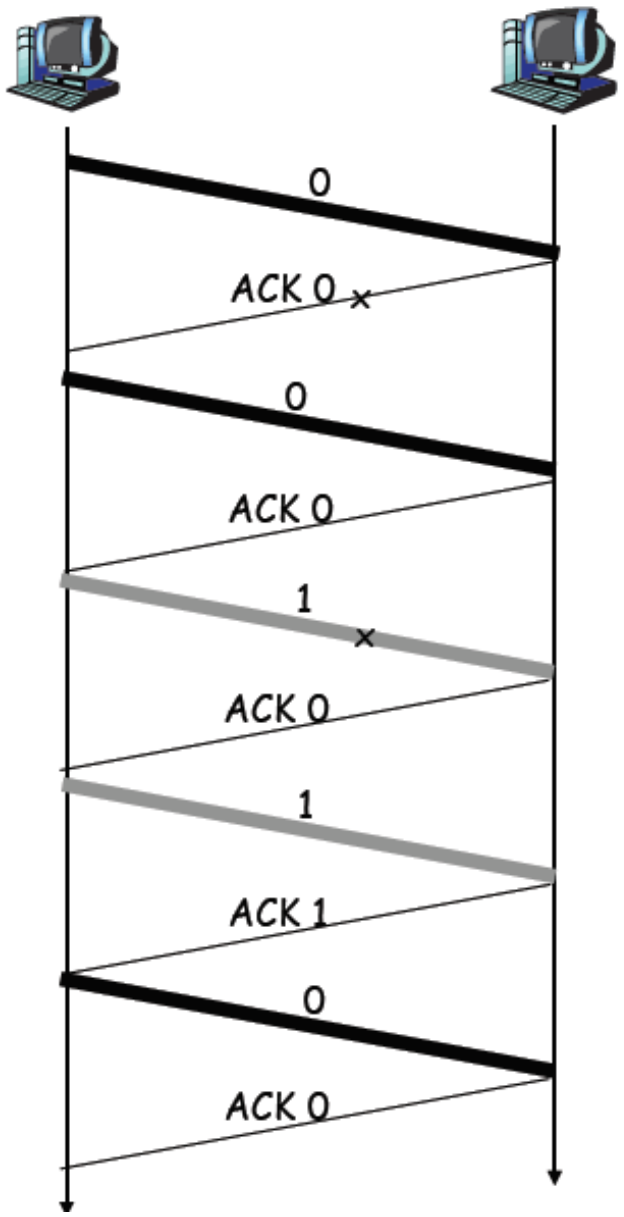
Stop & wait (u kanalu bez gubitaka)

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 posljednji ACK/NAK primljen ispravan od strane pošiljaoca



Stop & wait (u kanalu bez gubitaka) bez NAK

- Iste funkcionalnosti kao u prethodnom slučaju, korišćenjem samo ACK
- umjesto NAK, prijemnik šalje ACK za posljednji 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 (kanal sa greškom i gubicima)

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

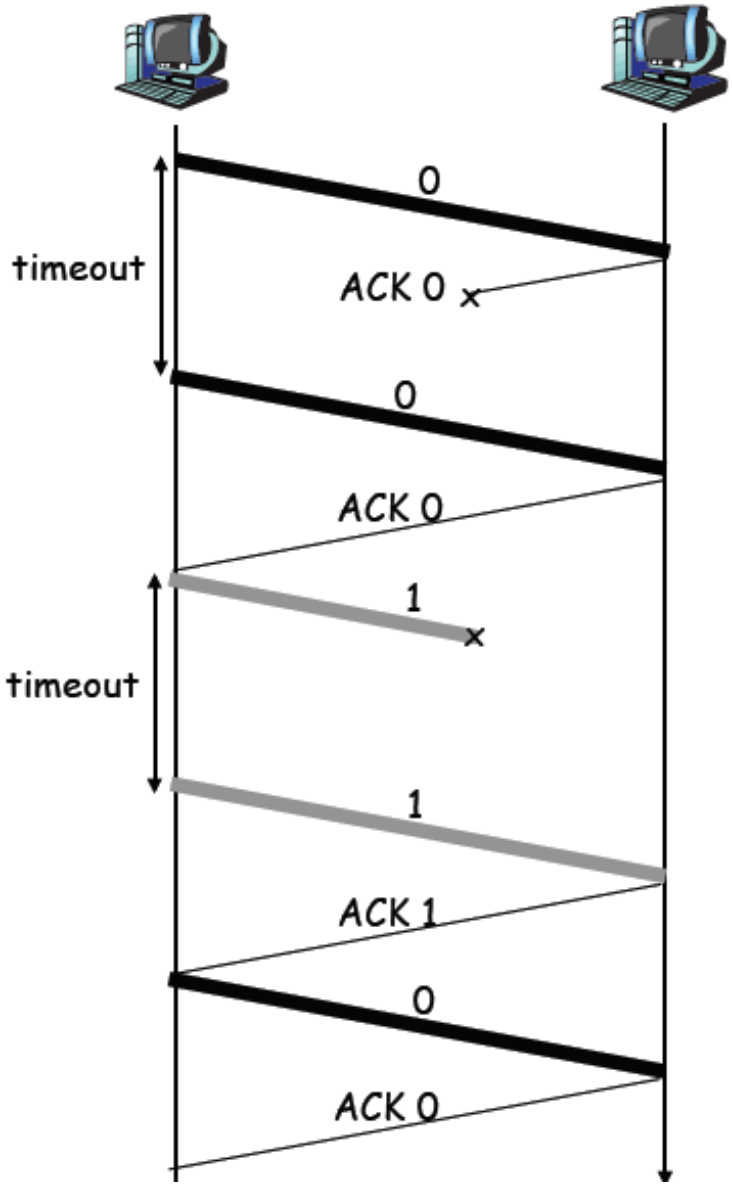
- checksum, broj u sekvenci, ACK, retransmisije su od pomoći, ali ne dovoljno.

P: 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?

Pristup: pošiljalac čeka "razumno" vrijeme za ACK

- Retransmisija se obavlja ako se ACK ne primi u tom vremenu
- Ako paket (ili ACK) samo zakasni (ne biva 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 posljednji paket 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 performanse

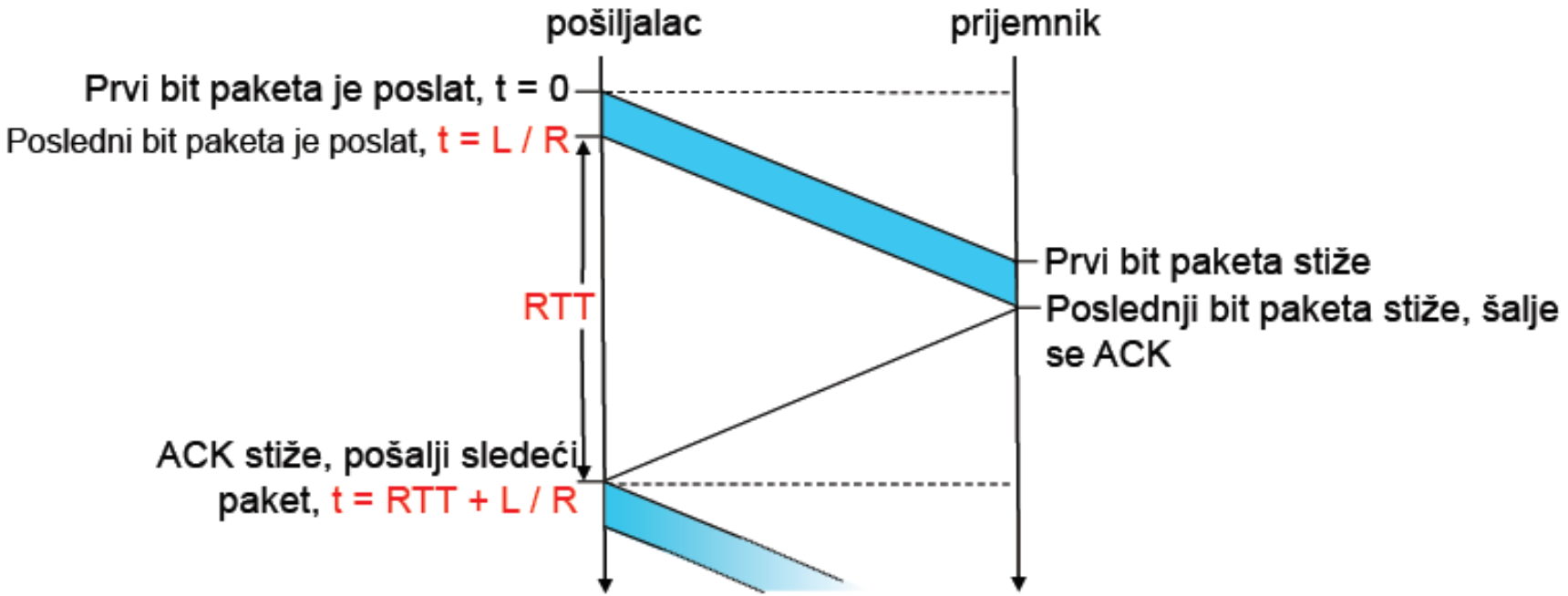
- S&W funkcioniše, ali ima loše performanse
- primjer: 1 Gb/s link, 15 ms vrijeme prenosa od kraja do kraja, veličina paketa 1000B :

$$T_{prenosa} = \frac{L\ (\text{veličina paketa u bitima})}{R\ (\text{propusnost linka, b/s})} = \frac{8kb/pkt}{10^9\ b/s} = 8\ \mu s$$

$$U_{pošilj.} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- $U_{pošiljalac}$: iskorišćenje – dio vremena u kome je pošiljalac zauzet
- Pošiljalac šalje 1000B paket svakih 30.008 ms -> 267kb/s bez obzira što je propusnost linka 1 Gb/s
- Mrežni protokol ograničava fizičke resurse!
- Stvar je još gora jer je napravljeno nekoliko zanemarivanja!

STOP & WAIT performanse

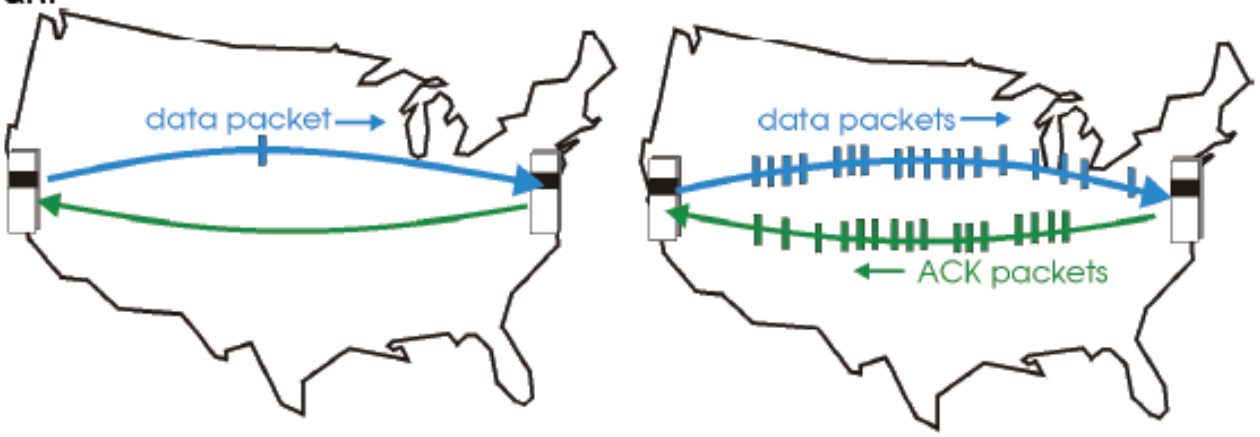


$$U_{\text{Pošilj.}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

"Pipelined" protokoli

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

- Opseg brojeva u sekvenci mora biti proširen
- Baferovanje više od jednog segmenta na predajnoj i/ili prijemnoj strani

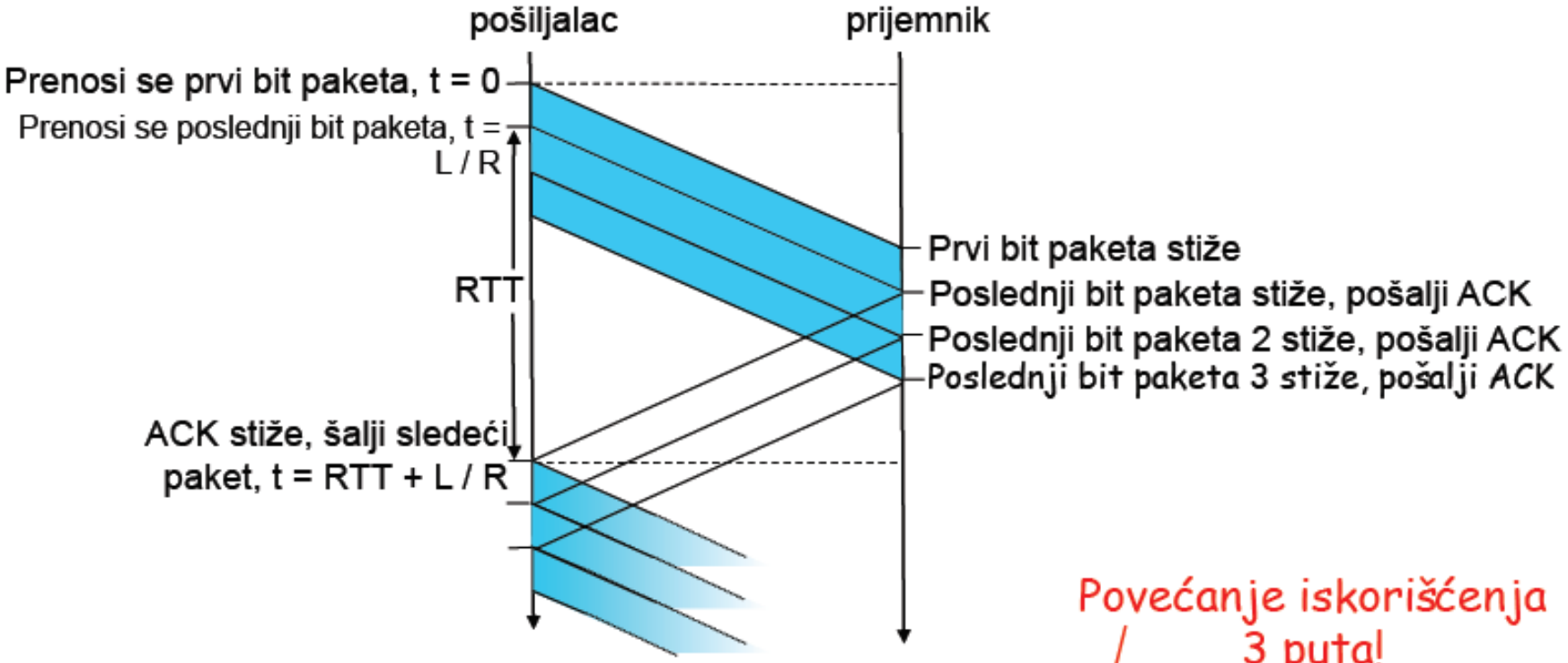


a) Stop and wait protokol

b) Pipeline protokol

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

"Pipelining": povećanje iskorišćenja



Povećanje iskorišćenja 3 puta!

$$U_{\text{Pošilj.}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Pipelined protokoli: pregled

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

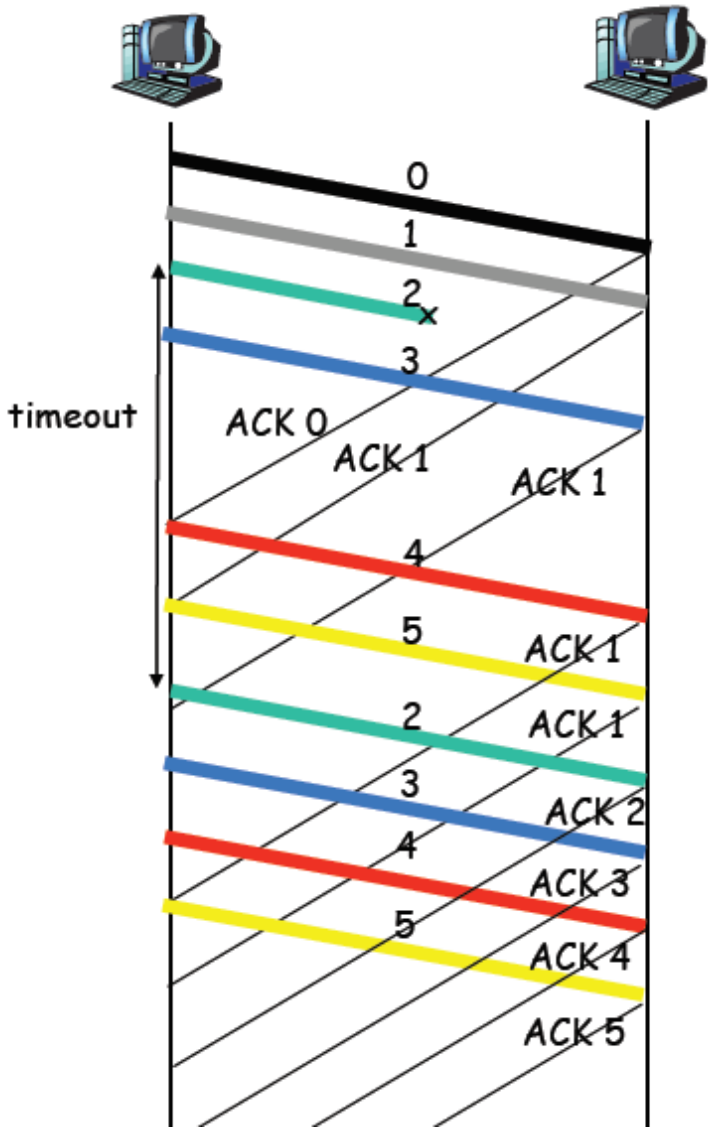
Selective Repeat:

- 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

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


0
1
2
3
timeout
ACK 0
ACK 1
ACK 1
4
5
ACK 1
2
ACK 1
3
ACK 2
4
ACK 3
5
ACK 4
ACK 5

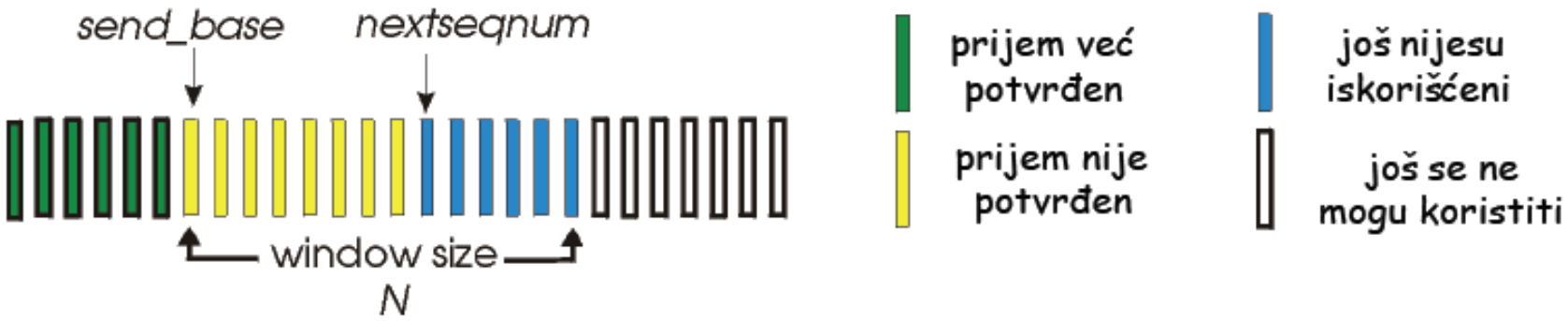
Go-Back-N: problemi

- 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 potpuno nepotrebno iznova poslati.
- Iz tog razloga se koriste "selective repeat" protokoli, koji kao što im ime kaže omogućavaju izbor paketa koji će biti ponovo poslati.

Go-Back-N (sliding window)

Pošiljalac:

- k-bita dugačak broj u sekvenci u zaglavlju paketa što 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, pri čemu se ne broje segmenti, već bajtovi u bajt streamu.
- ACK(n): ACK sve pakete, uključujući n-ti u sekvenci - "kumulativniACK"
  - Mogu se pojaviti dupli ACKovi (vidi prijemnik)

"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

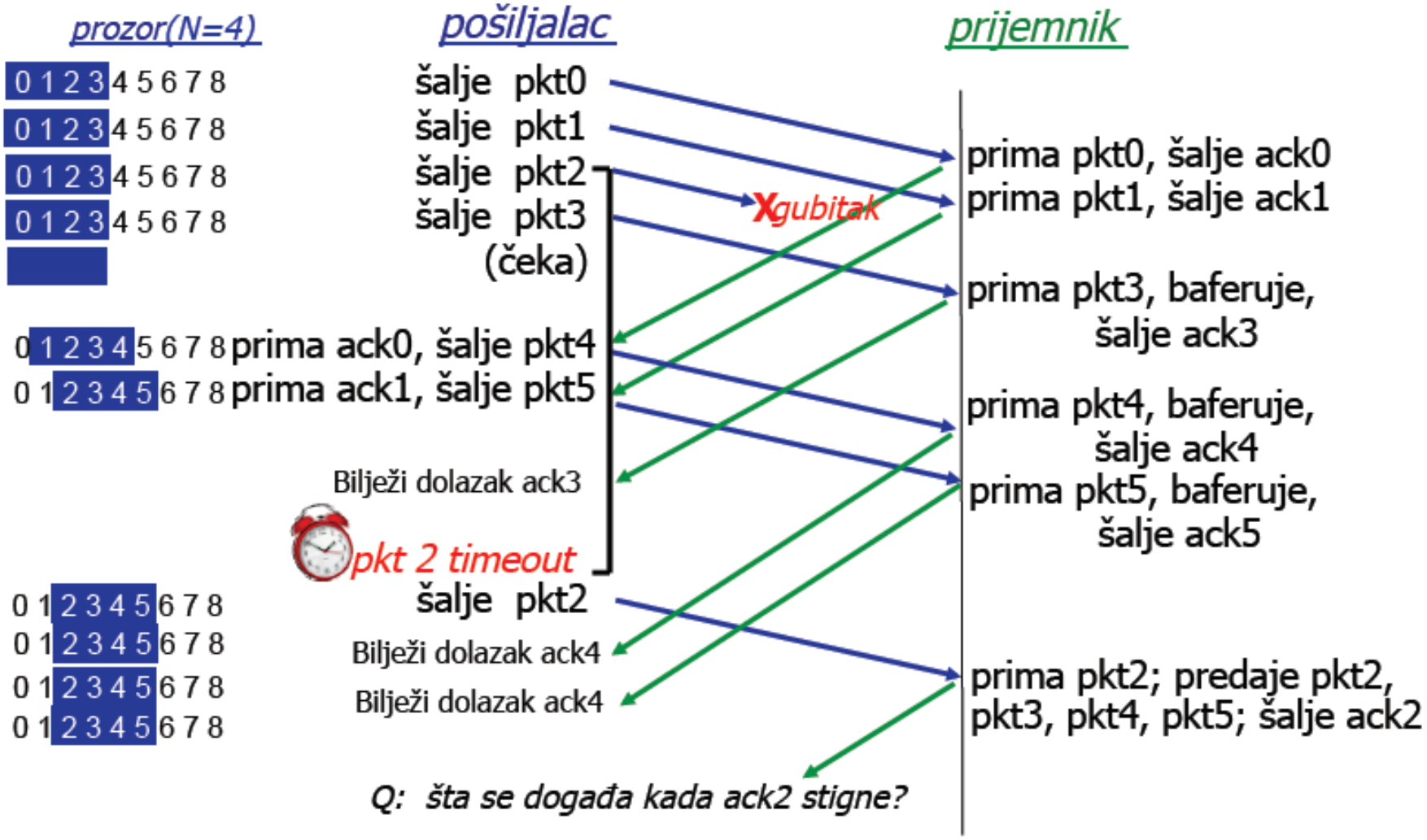
paket n u [rcvbase-N,rcvbase-1]

- ACK(n)

drugačije:

- ignoriši

Selective repeat u akciji

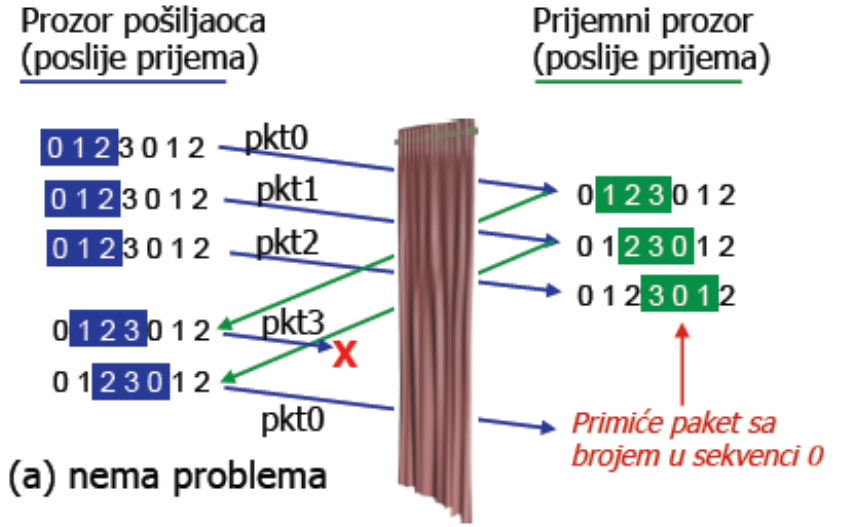


Selective repeat: dilema

primjer:

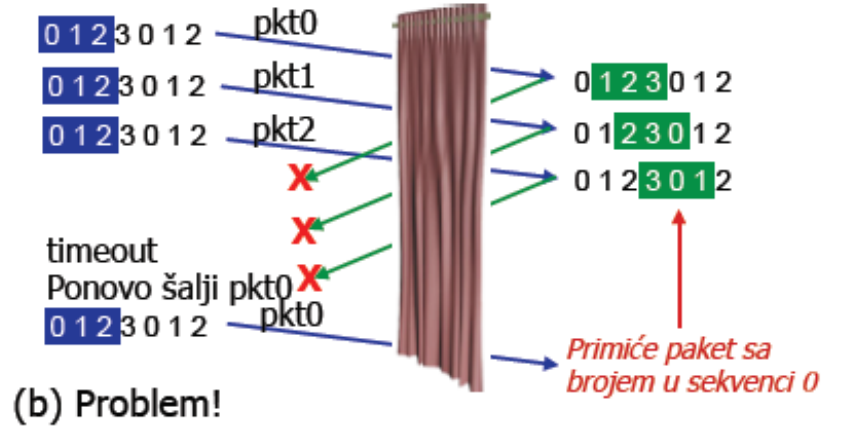
- Brojevi u sekvenci: 0, 1, 2, 3
- Veličina prozora=3
- Prijemnik ne vidi razliku u dva scenarija!
- Duplikat se prima kao novi (b)

Q: kakva je relacija između veličine broja u sekvenci i veličine prozora kako bi se izbjegao problem (b)?



Prijemnik ne vidi pošiljaočevu stranu.
Prijemnikovo ponašanje je identično u oba slučaja!

Nešto nije u redu!



TCP brojevi u sekvenci, ACK-ovi

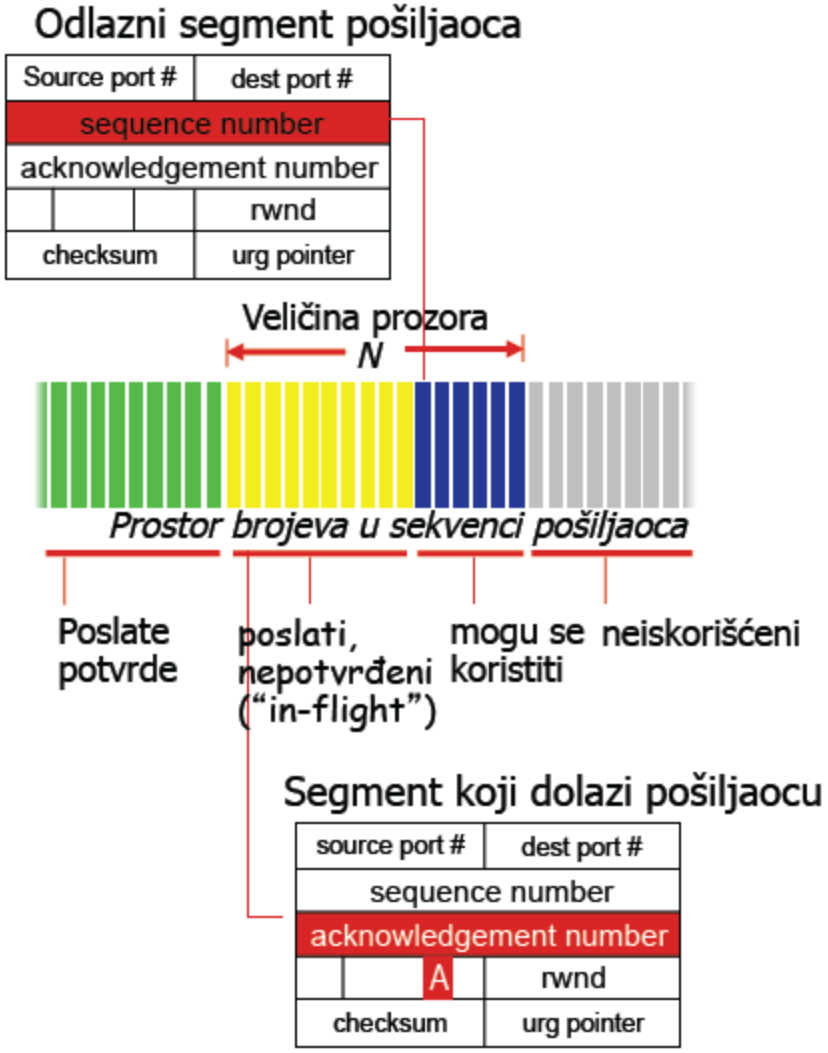
Brojevi u sekvenci:

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

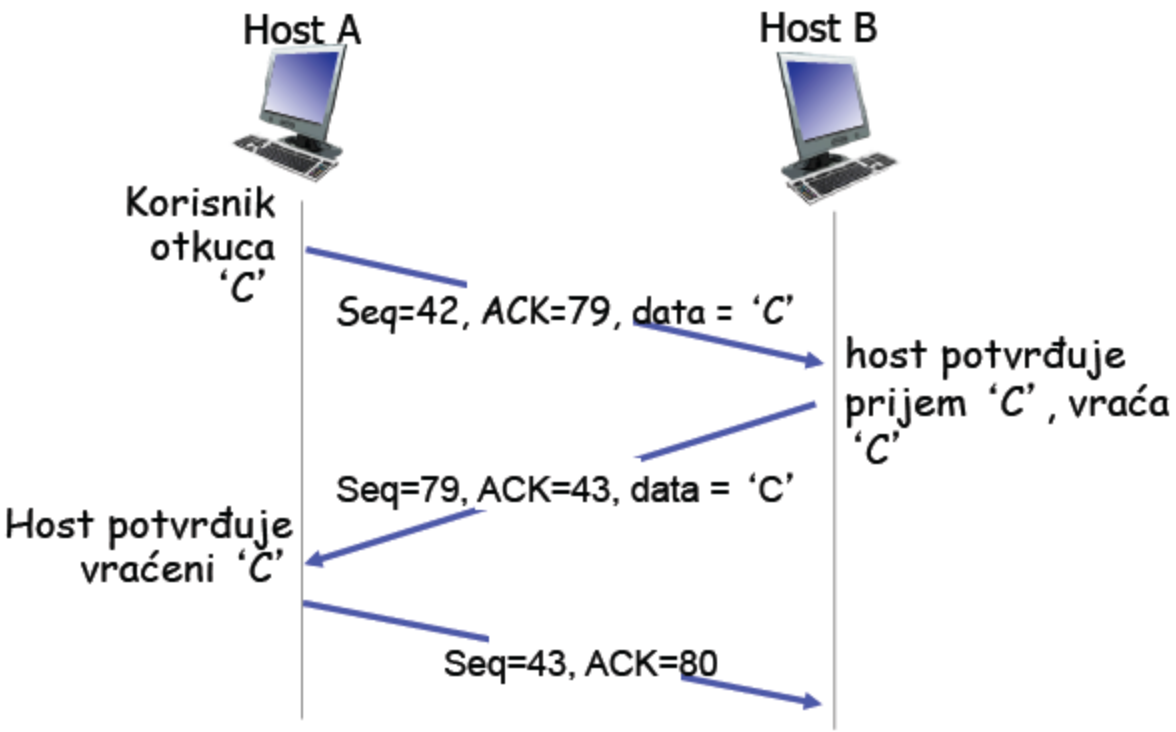
Potvrde (ACK):

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

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



TCP brojevi u sekvenci, potvrde



jednostavni Telnet scenario

TCP pouzdani prenos podataka

- TCP kreira rdt servis po IP nepouzdanom servisu
- "*Pipelined*" segmenti
- Kumulativne potvrde
- TCP koristi jedan retransmisioni tajmer

rdt - reliable data transfer

- Retransmisije su trigerovane 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 *byte*-stream broj prvog bajta podataka u segmentu
- Startuje se tajmer ako to već nije urađeno
- Interval *timeout*-a se izračunava po odgovarajućoj formuli

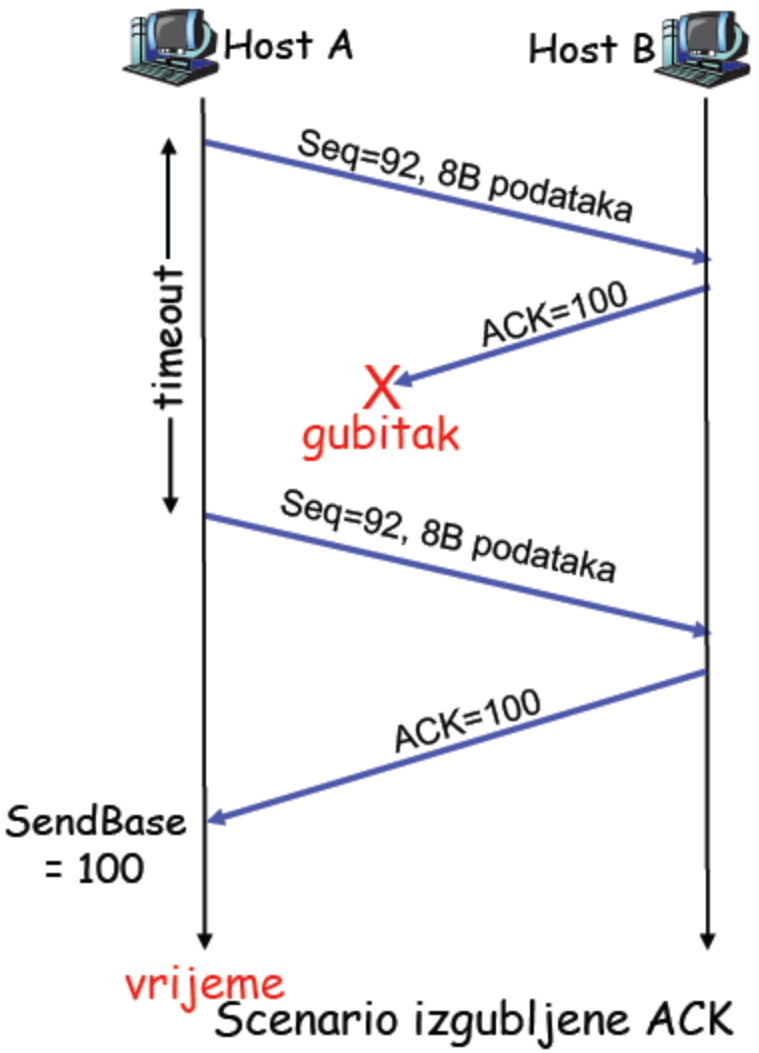
2. timeout:

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

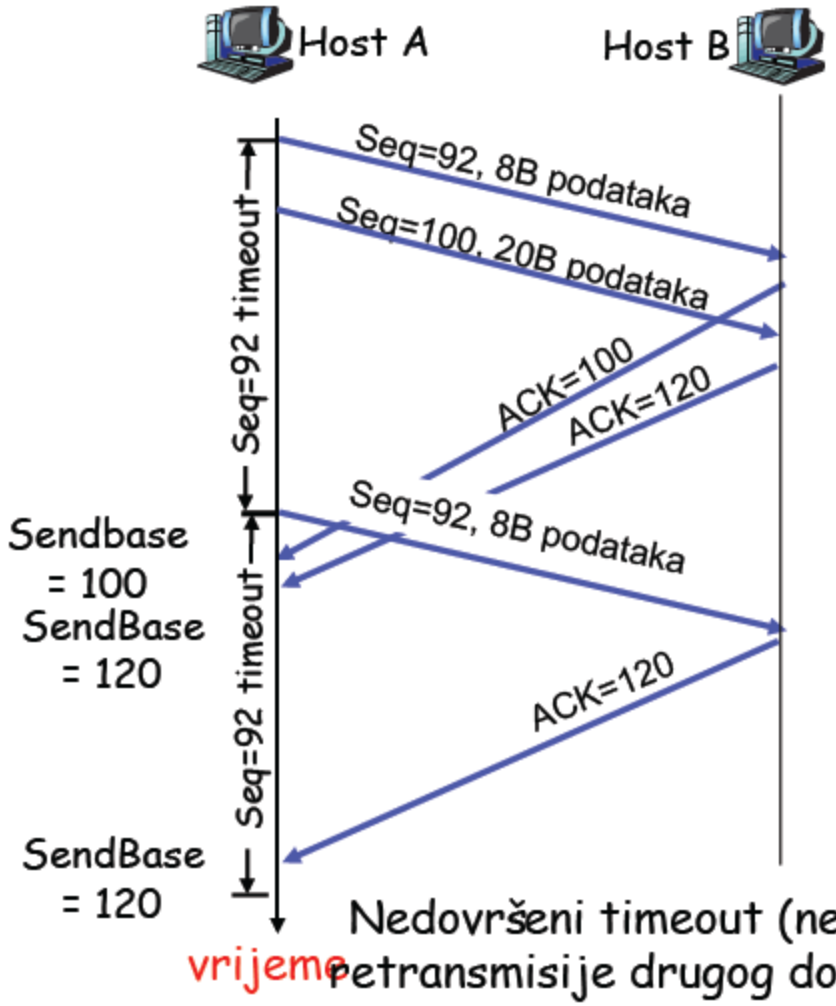
3.Ack primljen:

- Ako se potvrdi prijem ranije nepotvrđenog segmenta
  - Napraviti odgovarajući *update*
  - startovati tajmer ako postoje segmenti koji čekaju

TCP: scenariji retransmisije

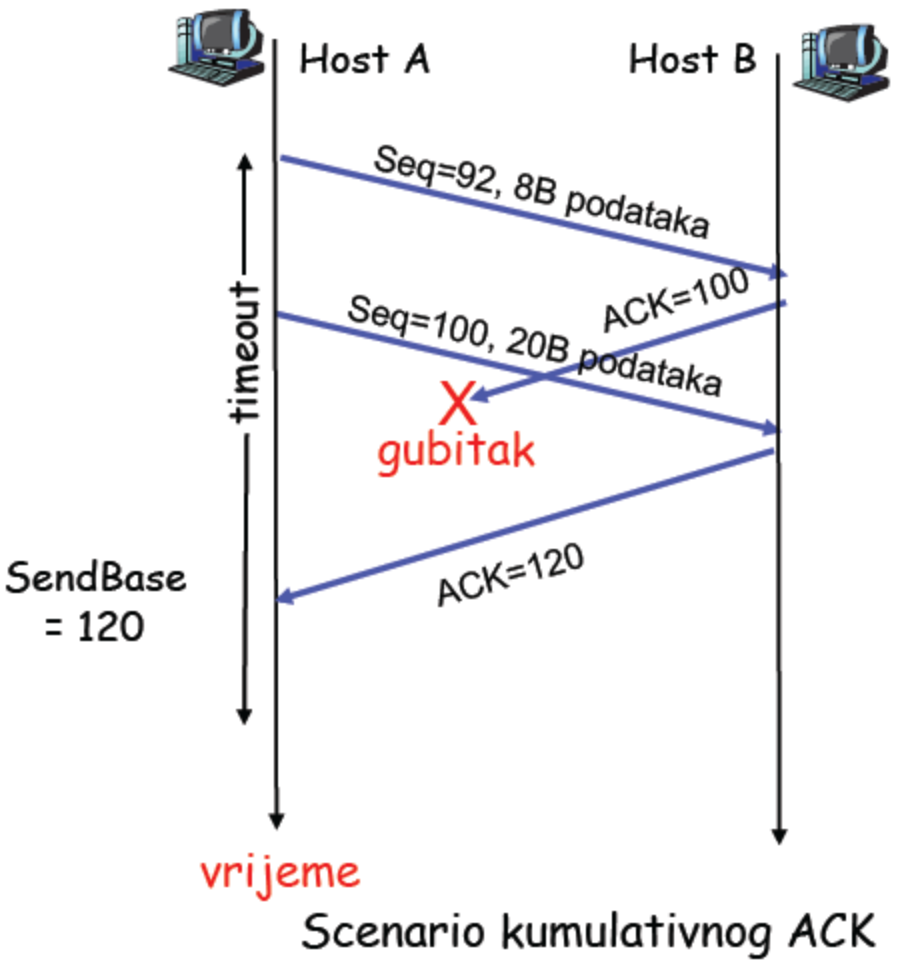


Scenario izgubljene ACK



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

TCP: scenariji retransmisije



Scenario kumulativnog ACK

U slučaju kada istekne *timeout* period, TCP se više ne pridržava ranije pomenute formule za izračunavanje timeout intervala. Umjesto nje TCP duplira raniju vrijednost *timeout* intervala.

TCP Round Trip Time i Timeout

P: kako postavitiTCP 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

P: kako aproksimirati RTT?

- `SampleRTT`: mjeriti vrijeme od slanja segmenta do prijema *ACK*
  - Ignorisati 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 ignorisati retransmisije?

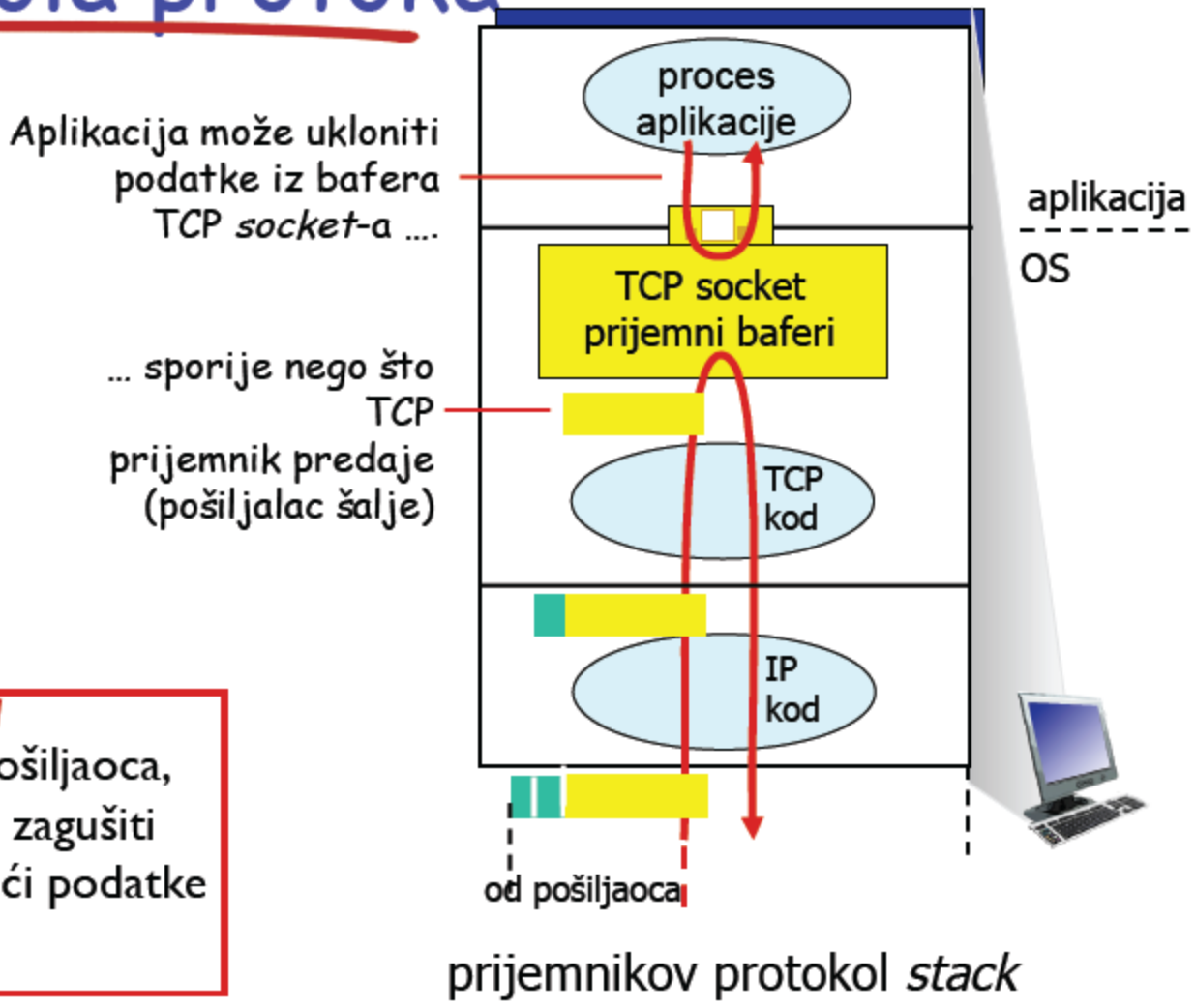
TCP generisanje ACK [RFC 1122, RFC 2581]

Događaj na prijemu	TCP akcije prijemnika
Dolazak in-order 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 in-order segmenta sa očekiv. brojem u sekvenci. Potvrđ. prijema drugog segmenta u toku.	Odmah šalje jednu kumulativnu ACK, potvrđujući oba in-order segmenta
Dolazak out-of-order 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

"Fast Retransmit"

- *Time out* period je često predug:
  - Dugo kašnjenje prije slanja 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 iste podatke, pretpostavlja se da je segment poslije potvrđenog izgubljen:
  - "fast retransmit": ponovno slanje segmenta prije nego što je tajmer istekao

TCP kontrola protoka

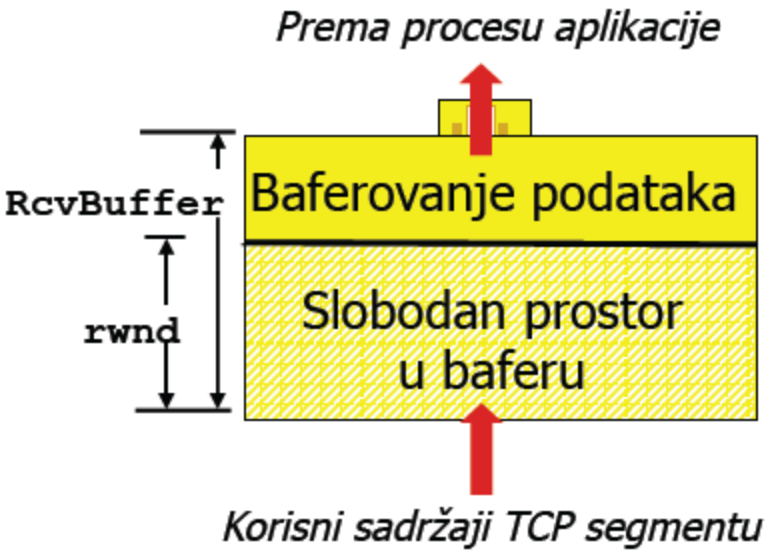


Kontrola protoka

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

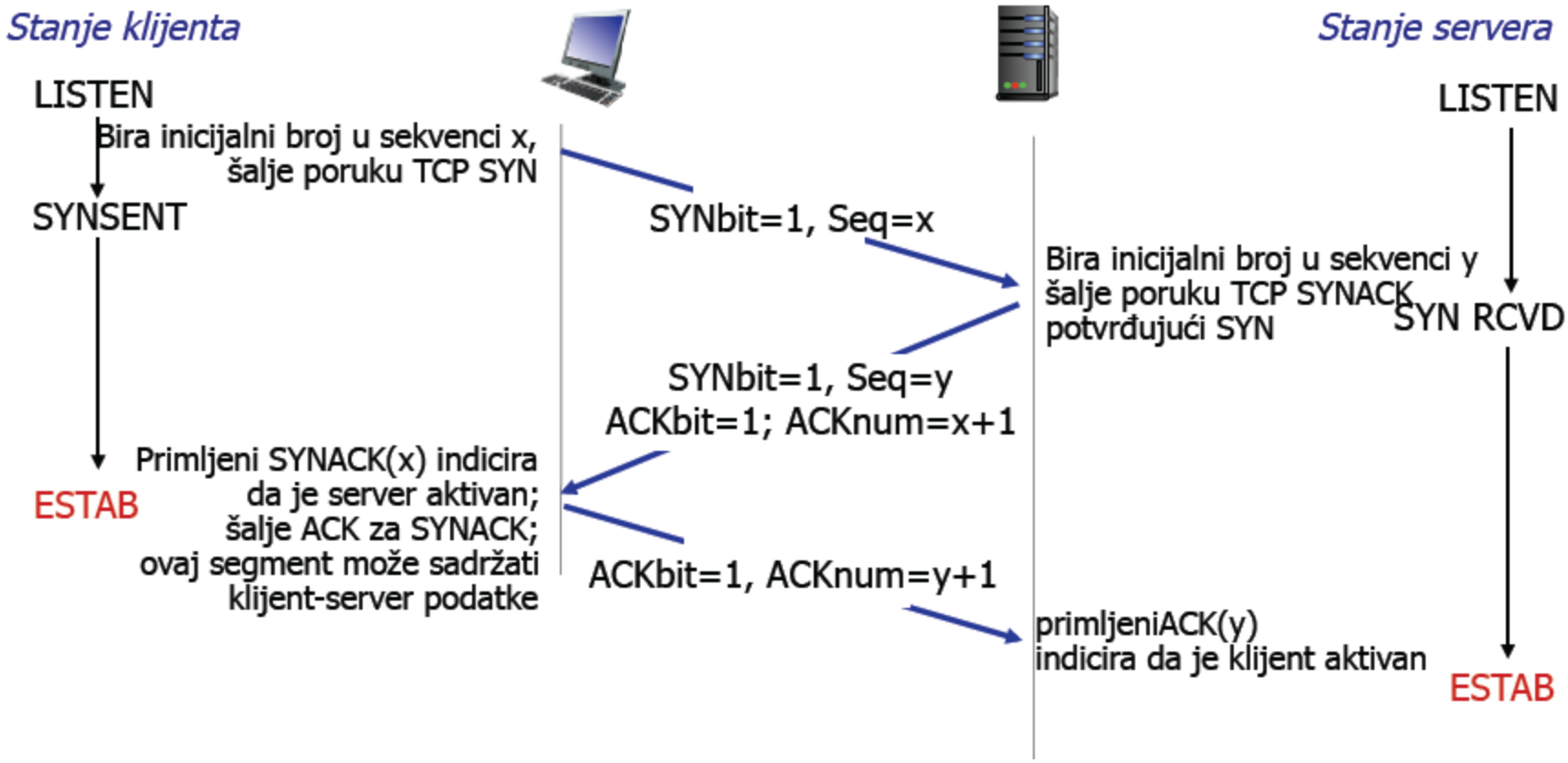
TCP flow control

- Prijemnik oglašava slobodan prostor u baferu podešavanjem vrijednosti u polje **`rwnd`** u zaglavljuTCP 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

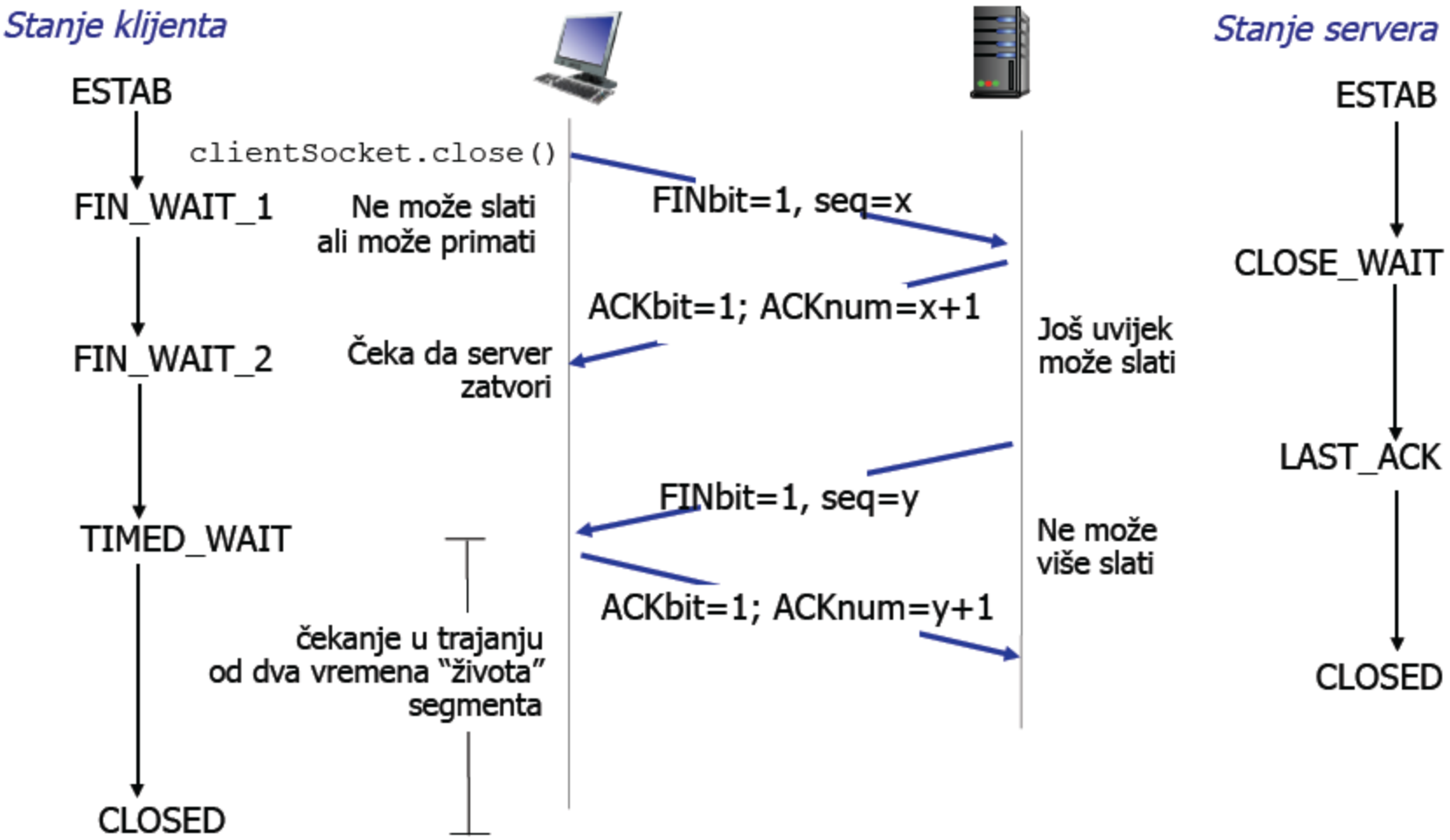


Baferovanje na prijemnikovoj strani

TCP 3-way handshake



TCP: zatvaranje konekcije



TCP kontrola zagušenja

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

  `LastByteSent-LastByteAcked`

  `≤ CongWin`

- Približno,

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

- **CongWin** je dinamička funkcija detekcije zagušenja mreže

Kako pošiljac otkriva zagušenje?

- gubitak = timeout *ili* 3 duplirane potvrde
- TCP pošiljalac smanjuje brzinu (**CongWin**) poslije gubitka

tri mehanizma:

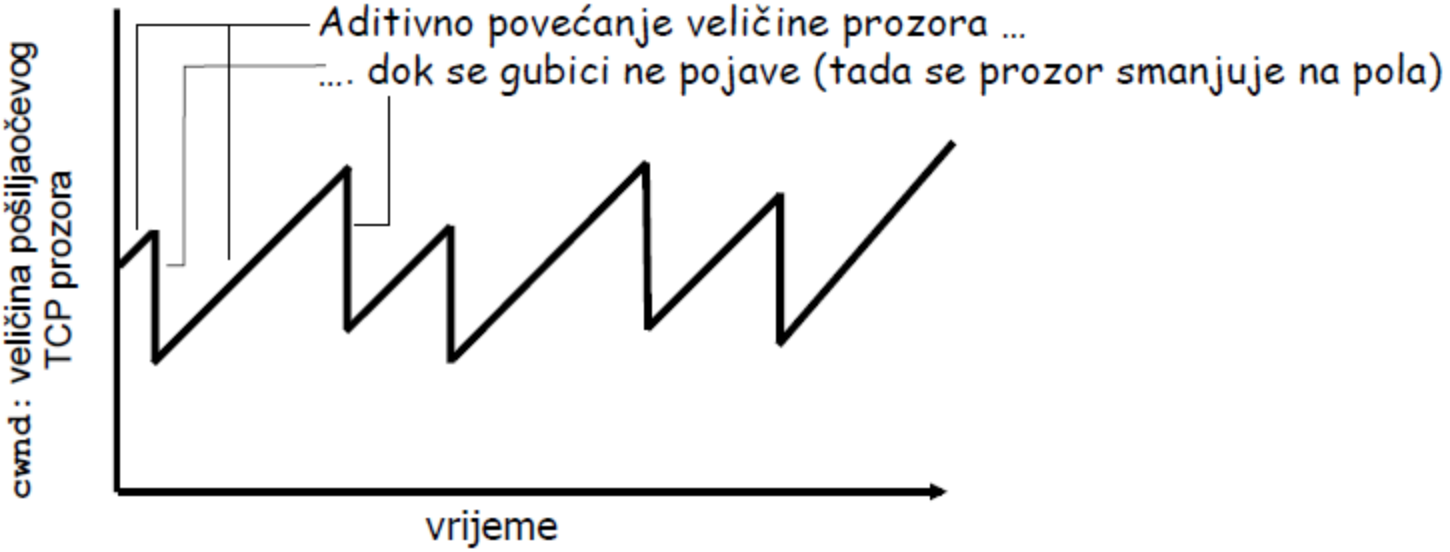
- AIMD
- "slow start"
- konzervativan poslije timeouta

AIMD - Additive increase/multiplicative decrease

TCP AIMD

Multiplikativno smanjenje: smanjuje **CongWin** na pola u slučaju gubitka

Aditivno povećanje: povećava **CongWin** za 1 MSS svaki RTT u odsustvu gubitka: *sondiranje*



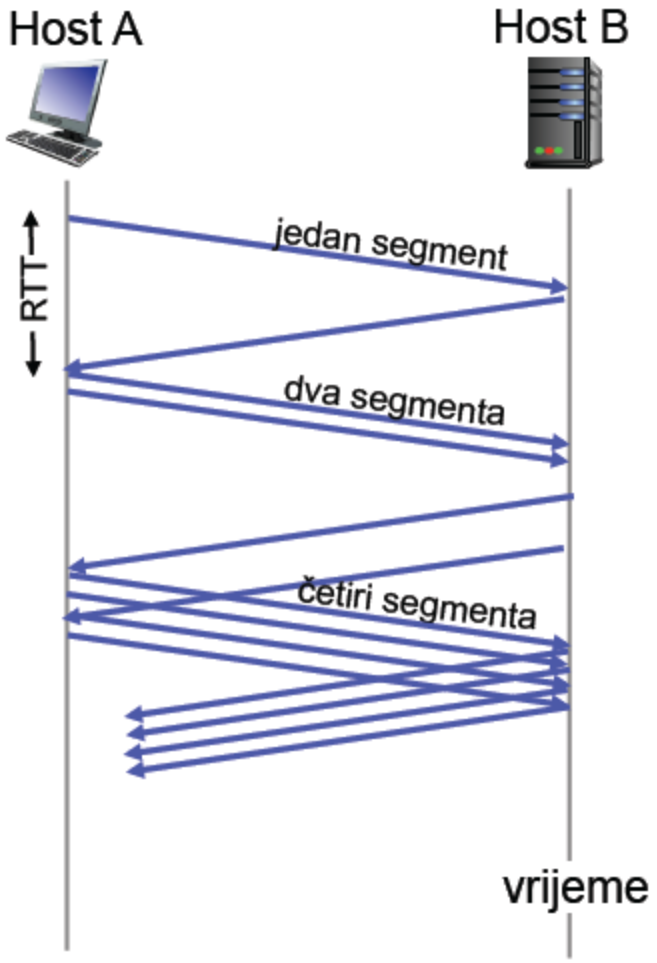
Duga TCP veza

TCP Slow Start

- Kada veza počne, `CongWin` = 1 MSS
  - Primjer: MSS = 500 B & RTT = 200 ms
  - Inicijalna brzina =20 kb/s
- Dostupna propusnost može biti>> MSS/RTT
  - Poželjno je brzo podešavaje na željenu brzinu
- Kada veza počne, povećava brzinu eksponencijalno do prvog gubitka

TCP Slow Start (više)

- Kada veza počne, eksponencijalno povećanje brzine do gubitka :
  - Udvostručuje se `CongWin` svaki RTT
  - Inkrementira se `CongWin` sa svakim primljenim
  - ACK Sumarum: inicijalna brzina je niska ali brzo raste



Ponavljanje

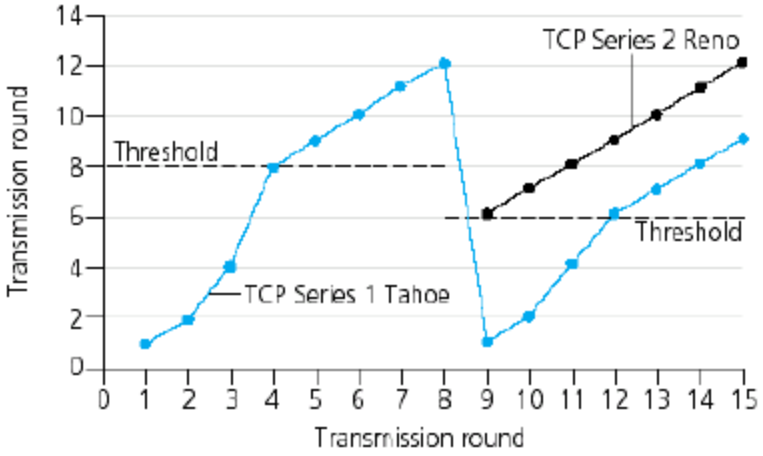
- Poslije 3 duplirane ACK:
  - `CongWin` se smanjuje na pola
  - Prozor raste linearno
- Ali posle timeout-a:
  - `CongWin` = 1 MSS;
  - Prozor raste eksponencijalno do praga a zatim linearno

Filozofija:

- 3 duple ACK indicira da je mreža sposobna da šalje
- timeout prije 3 duple ACK je "alarmantan"

Ponavljanje (više)

P: Kada eksponencijalna prelazi u linearnu?



Implementacija:

- Varijabilni prag (Tahoe)
- U slučaju gubitka, prag se postavlja na 1/2 vrijednosti CongWin prije gubitka
- U slučaju gubitka CongWin se smanjuje na pola(Reno)

TCP Tahoe

- "Slow Start", izbjegavanje kolizije
- Detektuje zagušenje kroz isticanje timeout-a i trostruke potvrde
- Inicijalizacija
  - CongWin=1;
  - Threshold=1/2 Max(Win)
- Poslije timeouta i trostruke potvrde
  - Threshold= 1/2 CongWin, CongWin= 1
  - Ulazi u slow start

TCP Reno

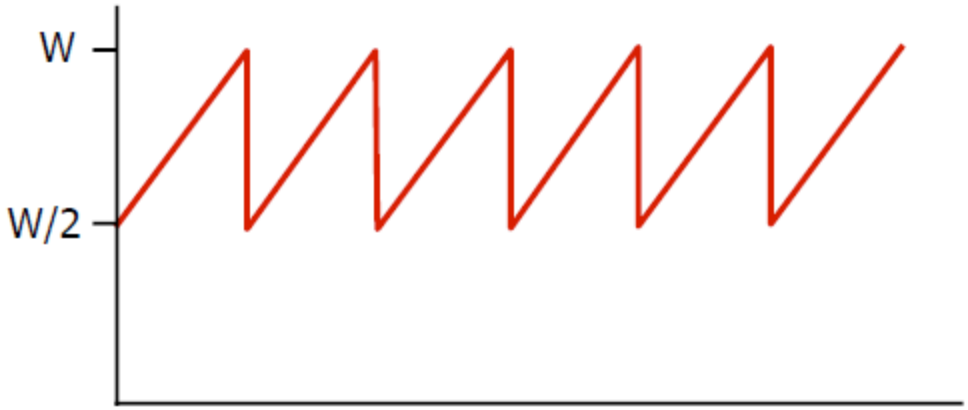
- "Fast Retransmit", "Fast recovery"
- Detektuje zagušenje kroz timeout-e i duplikate ACK-ova
- Kada se primi trostruki duplikat nekog ACK
  - Izbjegava slow start i ide direktno u fazu izbjegavanja kolizije
  - Threshold = 1/2 CongWin; Congwin = Threshold
    (Koristi AIMD)
- Kada se pojavi timeout
  - "Slow start"

Sumarum: TCP kontrola zagušenja

- Kada je **CongWin** ispod **Threshold**, pošiljalac je u slow-start fazi, prozor raste eksponencijalno.
- Kada je **CongWin** iznad **Threshold**, pošiljalac je u fazi izbjegavanja kolizije, prozor raste linearno.
- Kada se trostruki duplirani ACK pojavi, **Threshold** se setuje **CongWin/2** a **CongWin** se setuje na **Threshold**.
- Kada se pojavi timeout, **Threshold** se setuje na **CongWin/2** i **CongWin** se setuje na 1 MSS.

TCP propusnost

- 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



Budućnost TCP-a

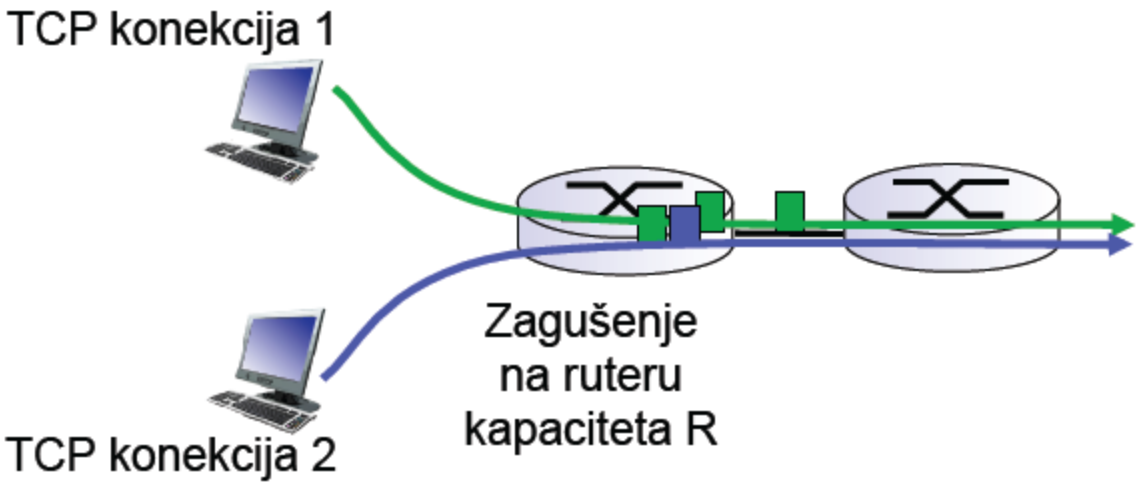
- primjer: 1500 B segmenti, 100ms RTT, želi se 10 Gb/s propusnost
- Zahtijeva se veličina prozora od W = 83,333 segmenata
- Srednja propusnost u zavisnosti vjerovatnoće gubitka:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- Vjerovatnoća gubitka ➜ $L = 2 \cdot 10^{-10}$
- Potrebne su nove verzije TCP-a za high-speed potrebe!
- http://netlab.caltech.edu/FAST/

Korektnost TCP

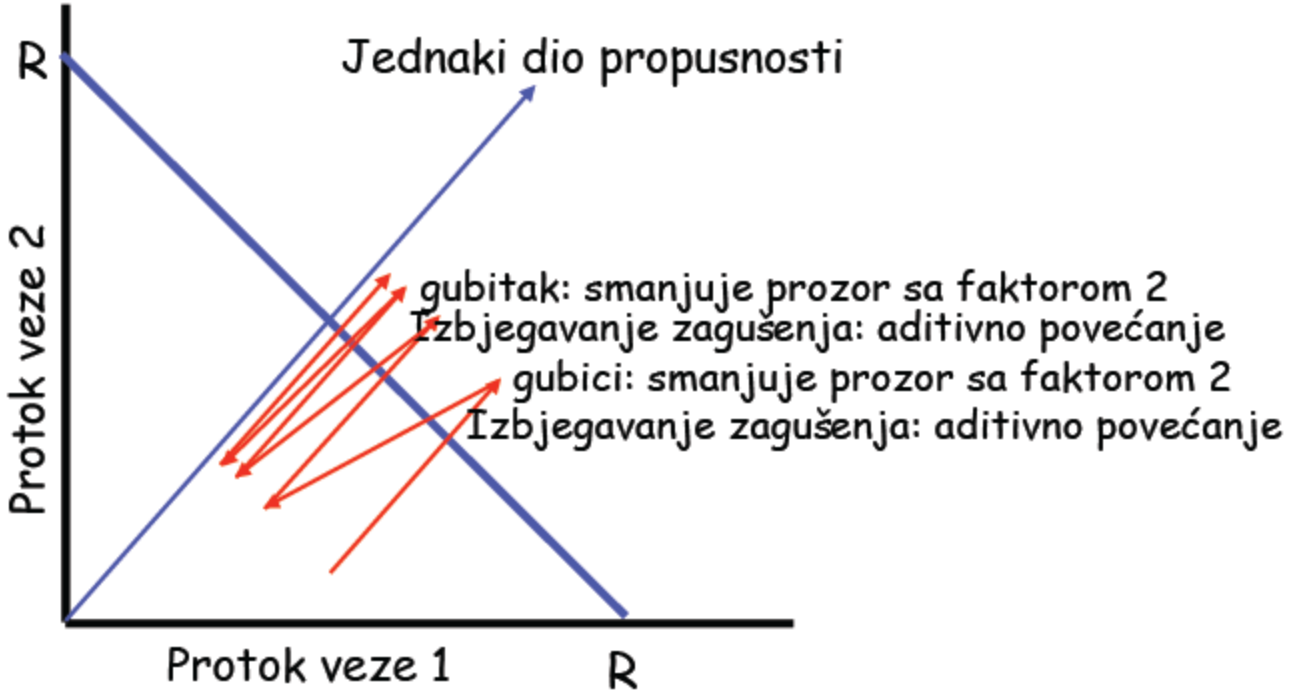
Cilj korektnosti: ako K TCP sesija dijele isti zagušeni link propusnosti R, svaki bi trebao da ima srednju propusnost od R/K



Zašto je TCP korektan?

Dvije sučeljene sesije:

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



Korektnost (više)

Korektnost i UDP

- Multimedijalne aplikacije često ne koristeTCP
  - Ne žele da kontrola zagušenja ograniči kapacitet
- Umjesto toga se koristi UDP:
  - Ubacuje audio/video konstantnom brzinom, toleriše gubitak paketa
- Oblast istraživanja: TCP

Korektnost i paralelne TCP konekcije

- Nema prevencije da aplikacija otvori paralelne veze između 2 hosta.
- Web browser-i to rade
- Primjer: link propusnosti R podržava 9 veza;
  - nova aplikacija pita za 1 TCP vezu, a dobija propusnost od R/10
  - Nova aplikacija pita za 11 novih TCP veza, i dobija više od R/2!