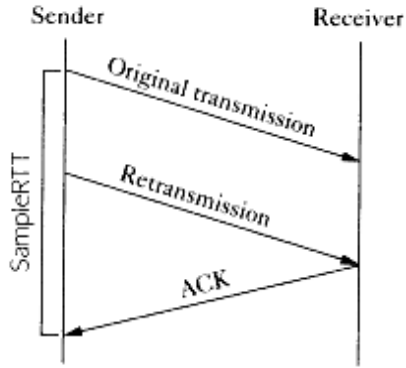
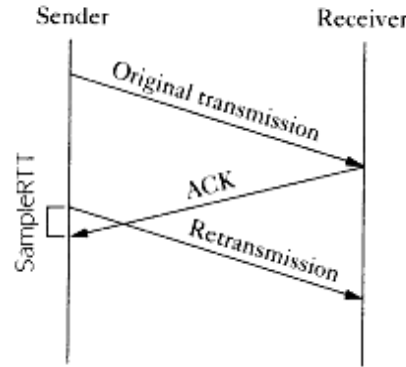


# 4. prednáška



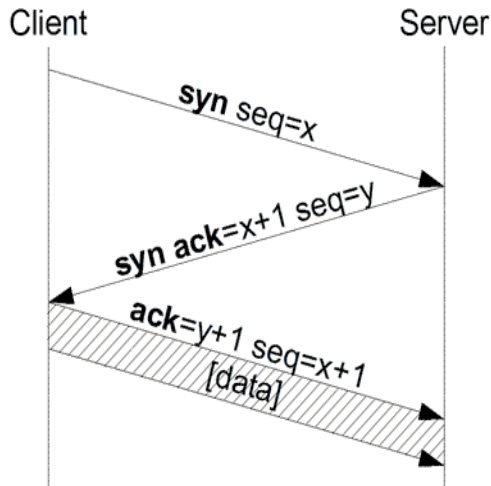
(a)



(b)

Source Port				Destination Port				
Sequence Number								
Acknowledgment Number								
Data Offset	Reserved	URG	ACK	PSH	RST	SYN	FIN	Window
Checksum				Urgent Pointer				
Options				Padding				

## Transportná vrstva



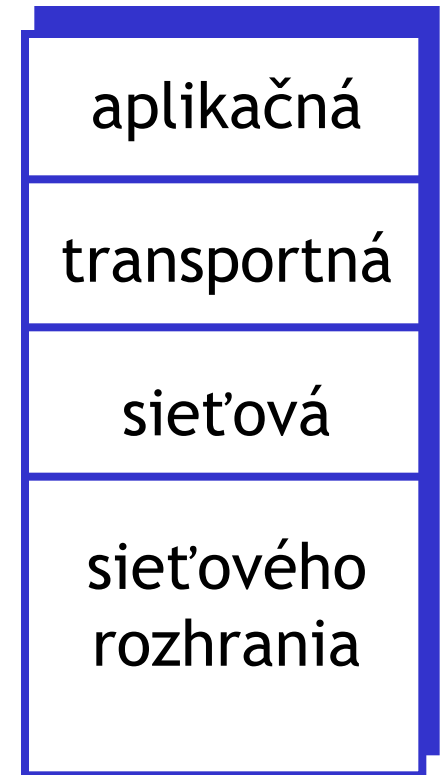
Source Port (16 bits)	Destination Port (16 bits)
Length (16 bits)	Checksum (16 bits)
Data....	

# Osnova rozprávania o transportnej vrstve

- ❑ 3.1 Služby transportnej vrstvy
- ❑ 3.2 Delenie správ a adresácia soketov
- ❑ 3.3 UDP: bezstavový transportný protokol
- ❑ 3.4 Princípy potvrdzovaného toku dát
- ❑ 3.5 TCP: stavový transportný protokol
  - ❖ pripájanie a odpájanie
  - ❖ štruktúra segmentu
  - ❖ potvrdzovaný tok dát
  - ❖ kontrola toku dát
- ❑ 3.6 Princípy zabezpečenia kontroly zahltenia
- ❑ 3.7 Kontrola zahltenia v protokole TCP

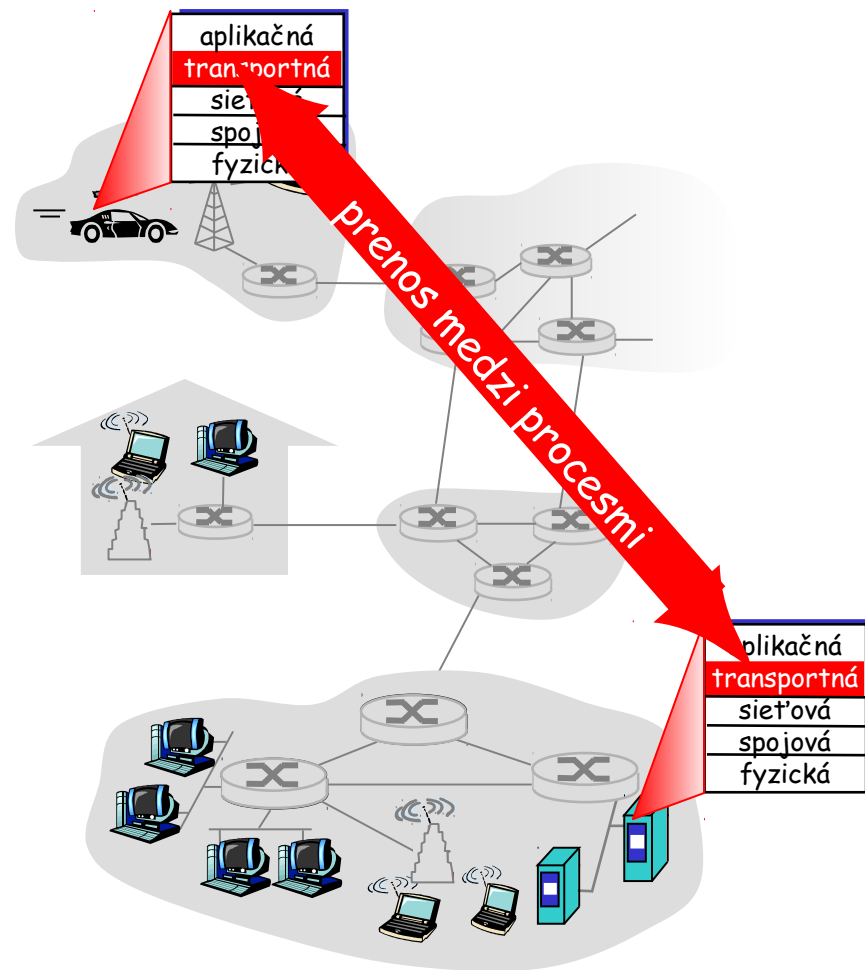
# Implementácia internetu TCP/IP

- ❑ **aplikačná (application):** umožňuje fungovanie sieťových aplikácií - definuje tvar a poradie správ
  - ❖ prezentačná a relačná splynuli s aplikačnou
    - tieto služby musí aj tak mať implementované aplikácia, ak to potrebuje
    - a čo ak nepotrebuje?
  - ❖ HTTP, FTP, SMTP, POP, IMAP, XMPP, SSH, ...
- ❑ **transportná (transport):** prenáša dáta medzi dvoma procesmi na rôznych koncových zariadeniach
  - ❖ TCP, UDP
- ❑ **sieťová (network):** smeruje datagramy od odosielateľa k príjemcovi
  - ❖ IP, smerovacie protokoly
- ❑ **sieťového rozhrania (network interface):** splynutie funkcionality do technológií na prenos dát medzi susednými sieťovými prvkami a spôsobu prenášania binárnych dát
  - ❖ PPP, Ethernet



# Protokoly a služby transportnej vrstvy

- poskytujú logický prenos dát medzi procesmi programov bežiacich na vzdialených koncových zariadeniach
- transportné protokoly sa realizujú iba na koncových stanicach
  - ❖ **odosielajúca strana: rozdeľuje správy** odosielané z procesu a vkladá ich do **segmentov** a tie posiela na spracovanie sieťovej vrstve
  - ❖ **prijímajúca strana: spája** dáta zo **segmentov** prijaté zo sieťovej vrstvy **do správ** a posiela ich procesu v aplikačnej vrstve
- je niekoľko transportných protokolov:
  - ❖ štandardne na internete: TCP a UDP



# Transportná verzus sieťová vrstva

- *transportná vrstva*: umožňuje logickú komunikáciu medzi procesmi
  - ❖ spolieha sa na služby sieťovej vrstvy
- *sieťová vrstva*: umožňuje logickú komunikáciu medzi stanicami
  - ❖ transportná vrstva iba povie, ktorej cieľovej stanici je segment určený

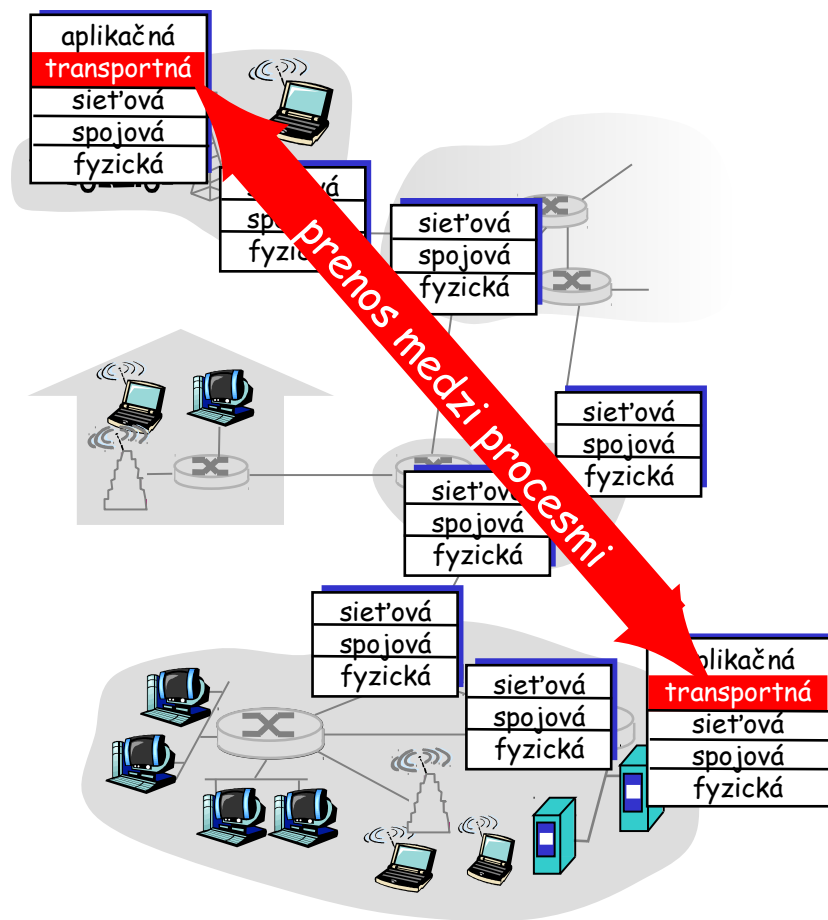
## Analógia “Koncert Madonny”

*Treba dopraviť pódium a techniku z Londýna do Košíc*

- procesy = londýnsky a košický manažér
- správy aplikačnej vrstvy = pódium a technika
- koncové stanice = mestá
- transportný protokol = tímy technikov v koncových mestách a komunikácia medzi nimi
- protokol sieťovej vrstvy = zásielková služba (64 kamiónov, 2 lode a logistika)

# Internetové protokoly transportnej vrstvy

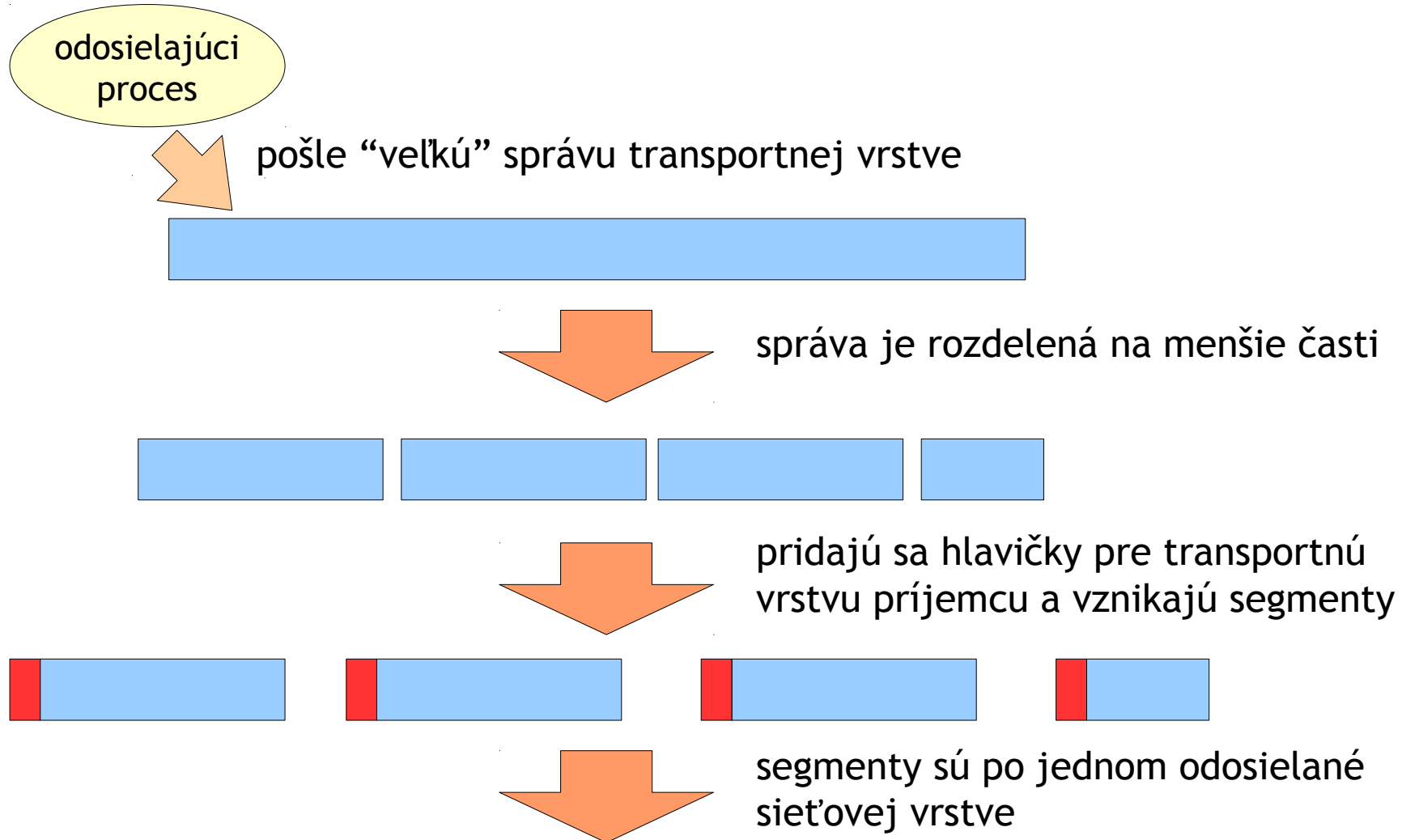
- potvrdzované doručenie v správnom poradí: TCP
  - ❖ potvrdzovanie segmentov
  - ❖ kontrola zahltenia
  - ❖ kontrola toku dát
  - ❖ nastavovanie spojenia
- nepotvrdzované doručenie v “náhodnom” poradí: UDP
  - ❖ žiadne extra vylepšenie oproti spoľahlivosti IP
- neposkytované služby:
  - ❖ zabezpečenie času doručenia
  - ❖ zabezpečenie prenosovej rýchlosti



# Osnova rozprávania o transportnej vrstve

- ❑ 3.1 Služby transportnej vrstvy
- ❑ 3.2 Delenie správ a adresácia soketov
- ❑ 3.3 UDP: bezstavový transportný protokol
- ❑ 3.4 Princípy potvrdzovaného toku dát
- ❑ 3.5 TCP: stavový transportný protokol
  - ❖ pripájanie a odpájanie
  - ❖ štruktúra segmentu
  - ❖ potvrdzovaný tok dát
  - ❖ kontrola toku dát
- ❑ 3.6 Princípy zabezpečenia kontroly zahltenia
- ❑ 3.7 Kontrola zahltenia v protokole TCP

# Delenie správ do segmentov





# Delenie a spájanie správ


## U odosielaťa:

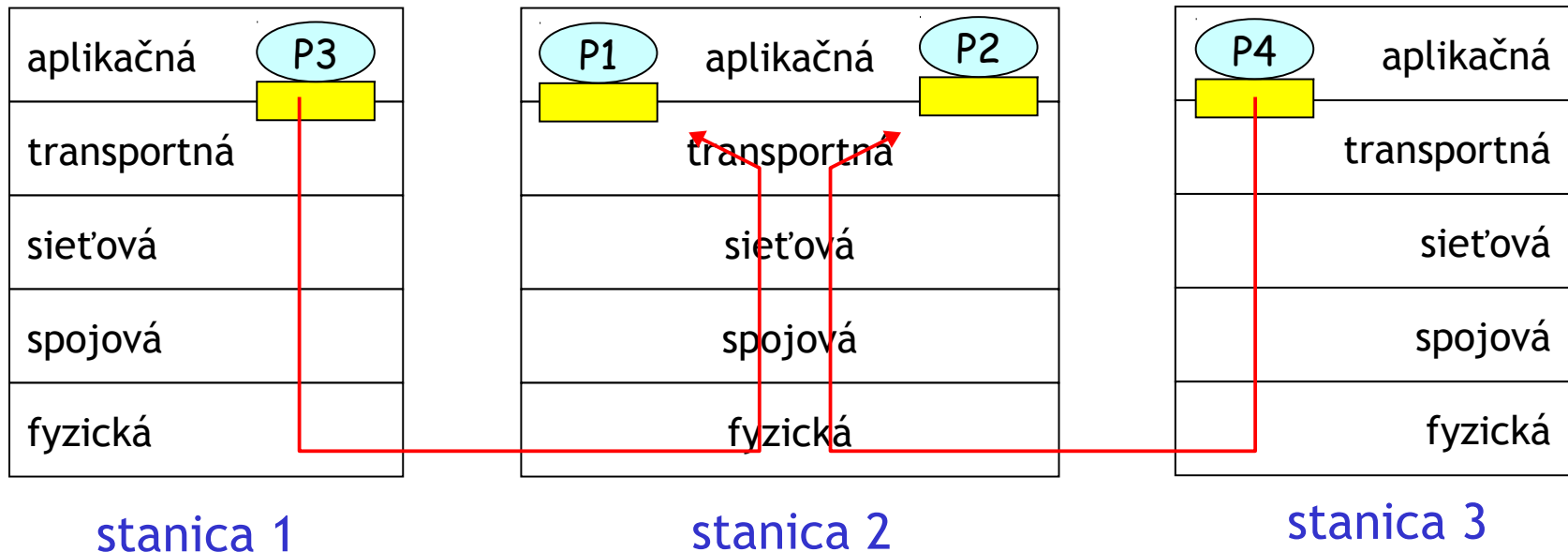
zobieranie správ zo soketov,  
vytvorenie segmentov  
a odoslanie príjemcovi

## U príjemcu:

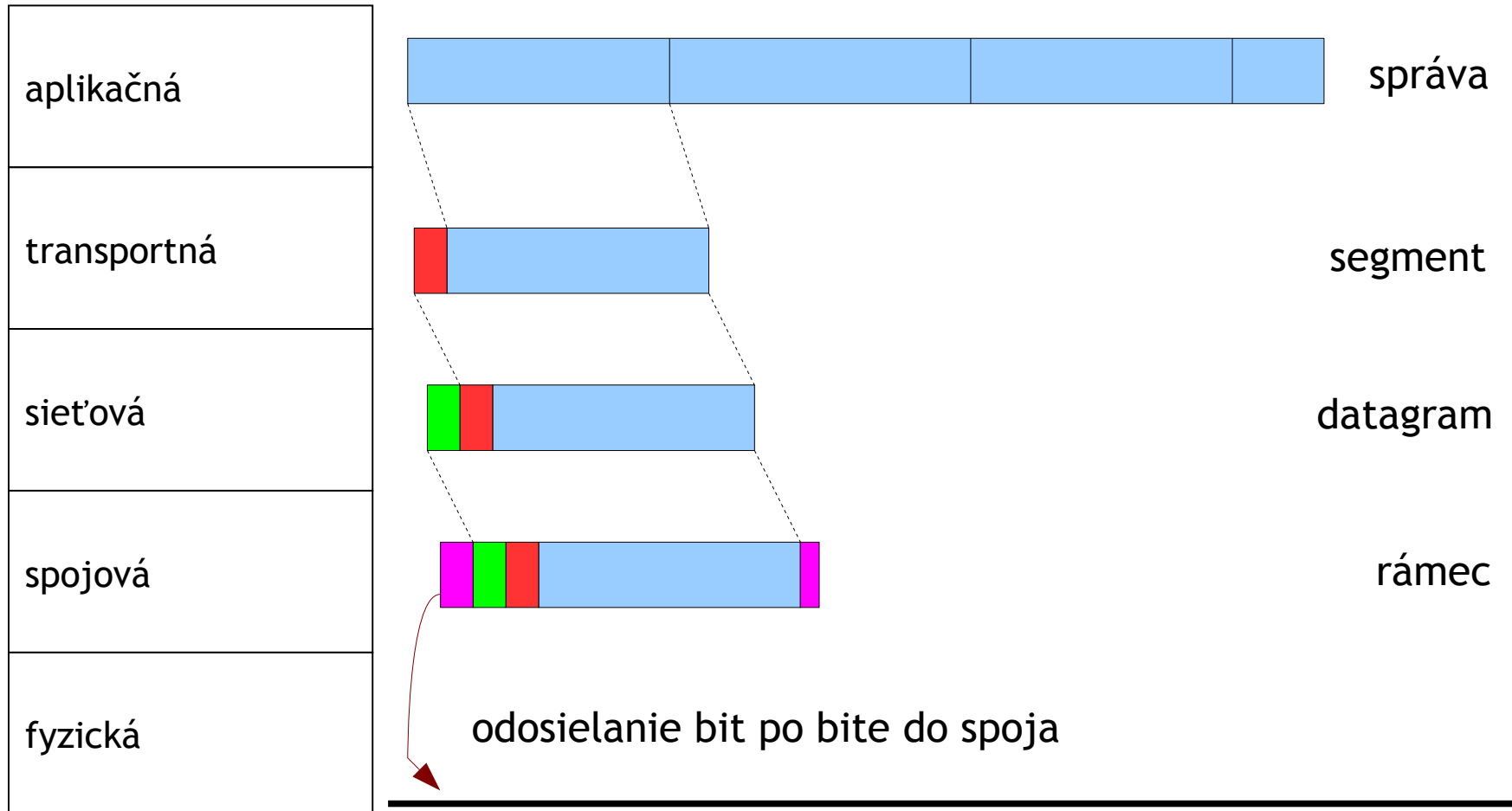
vyskladanie správ z prijatých  
segmentov a ich doručenie  
k správnym soketom

 = soket

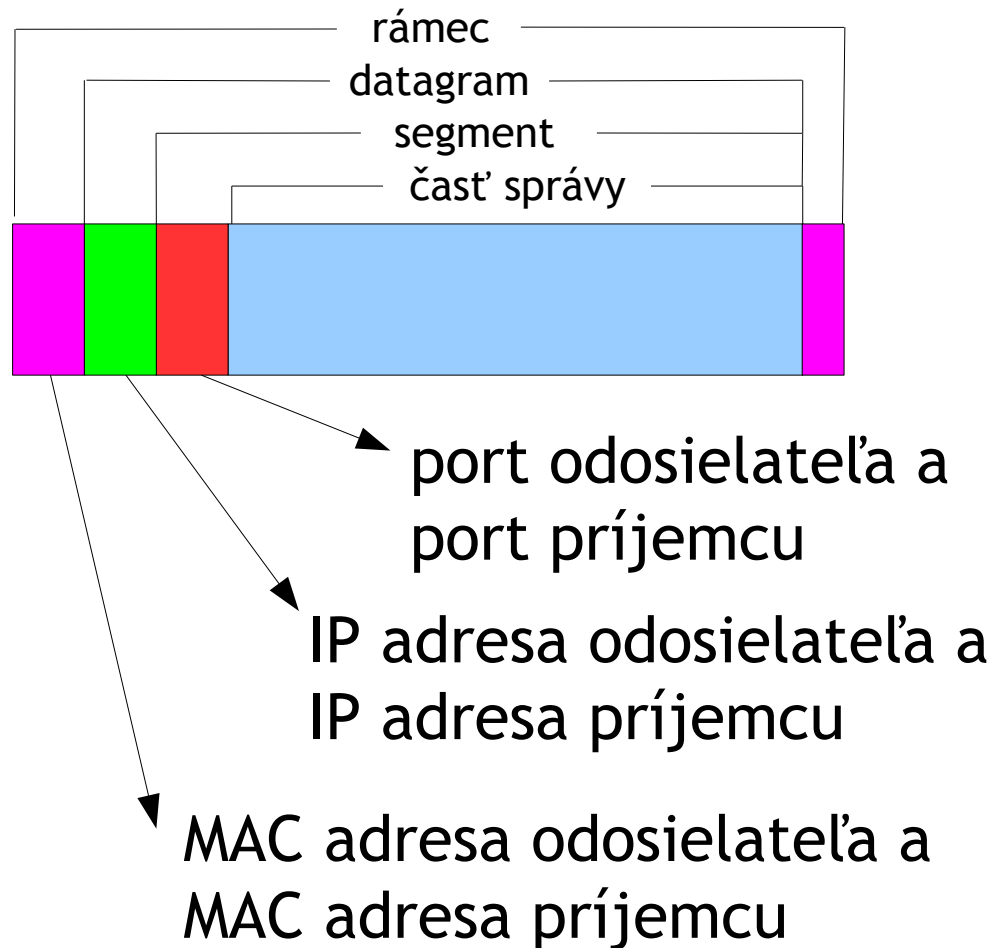
 = proces



# Tvorba paketu

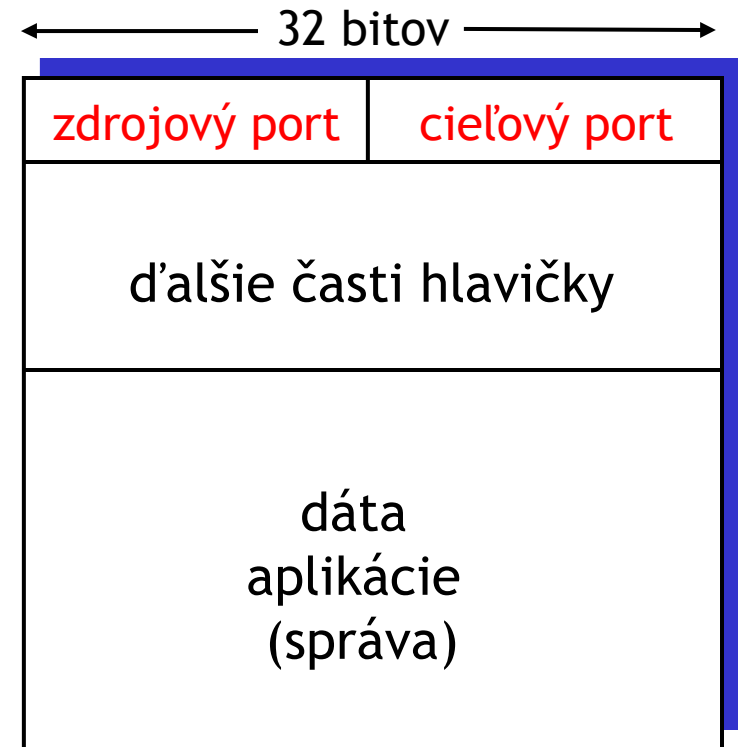


# Adresovacie údaje v pakete



# Odosielanie

- ❑ stanica dostane IP datagram
  - ❖ hlavička IP datagramu má zdrojovú IP adresu a cieľovú IP adresu
  - ❖ sieťová vrstva odovzdá z datagramu vyextrahovaný segment transportnej vrstve
  - ❖ hlavička segmentu má zdrojový port a cieľový port
- ❑ Transportná vrstva používa IP adresy a čísla portov na nasmerovanie segmentu k príslušnému soketu



formát segmentu TCP/UDP

# Adresácia soketov v UDP

- nový soket sa asociuje s číslom portu a prípadne aj s niektorou konkrétnou IP adresou stanice:

```
soket1 = new  
DatagramSocket(12534);  
soket2 = new  
DatagramSocket(12535,  
                "158.197.31.4");
```

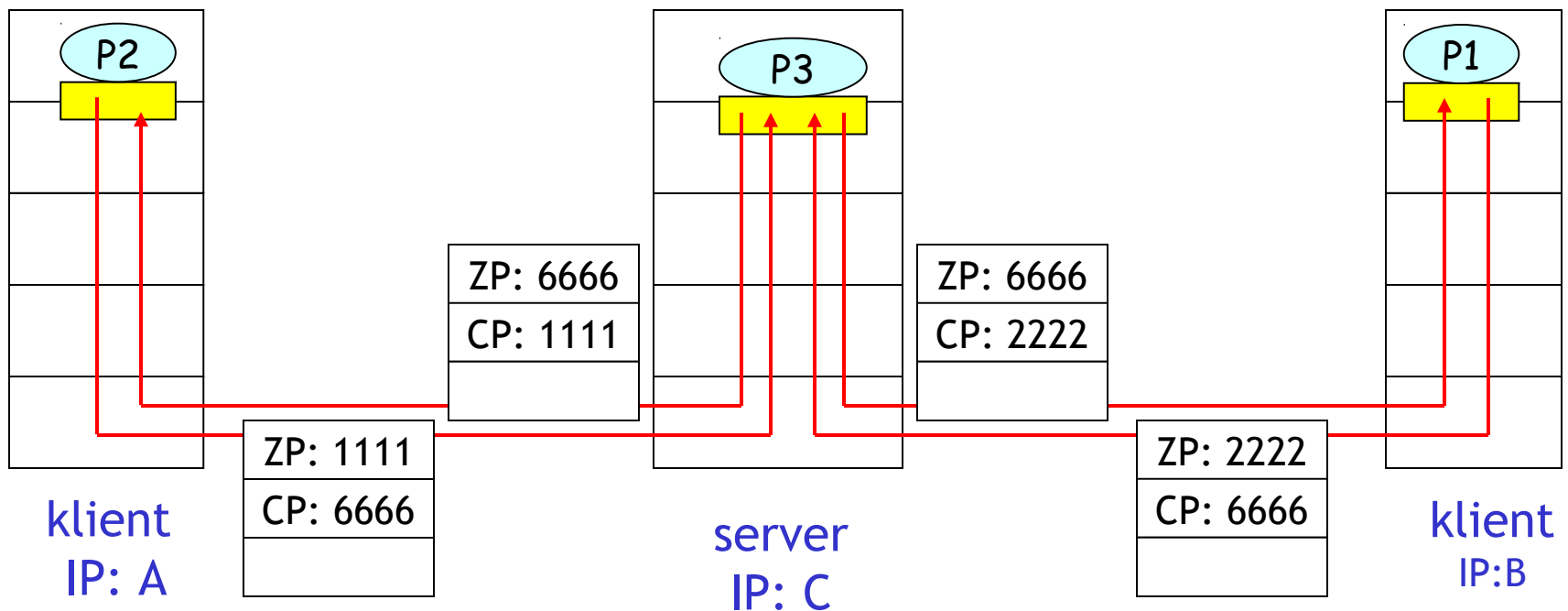
- UDP soket je jednoznačne identifikovaný dvojicou:  
(cieľová IP adresa, cieľový port)

- Keď stanica prijme UDP segment:
  - ❖ prečíta si číslo cieľového portu a cieľovú IP adresu
  - ❖ pošle UDP segment soketu asociovanému s týmto číslom portu
- pakety s rôznymi zdrojovými IP adresami a/alebo zdrojovými číslami portov môžu byť nasmerované k rovnakému soketu

# Adresácia soketov v UDP

■ = soket      ○ = proces

```
DatagramSocket serverSocket = new DatagramSocket(6666);
```



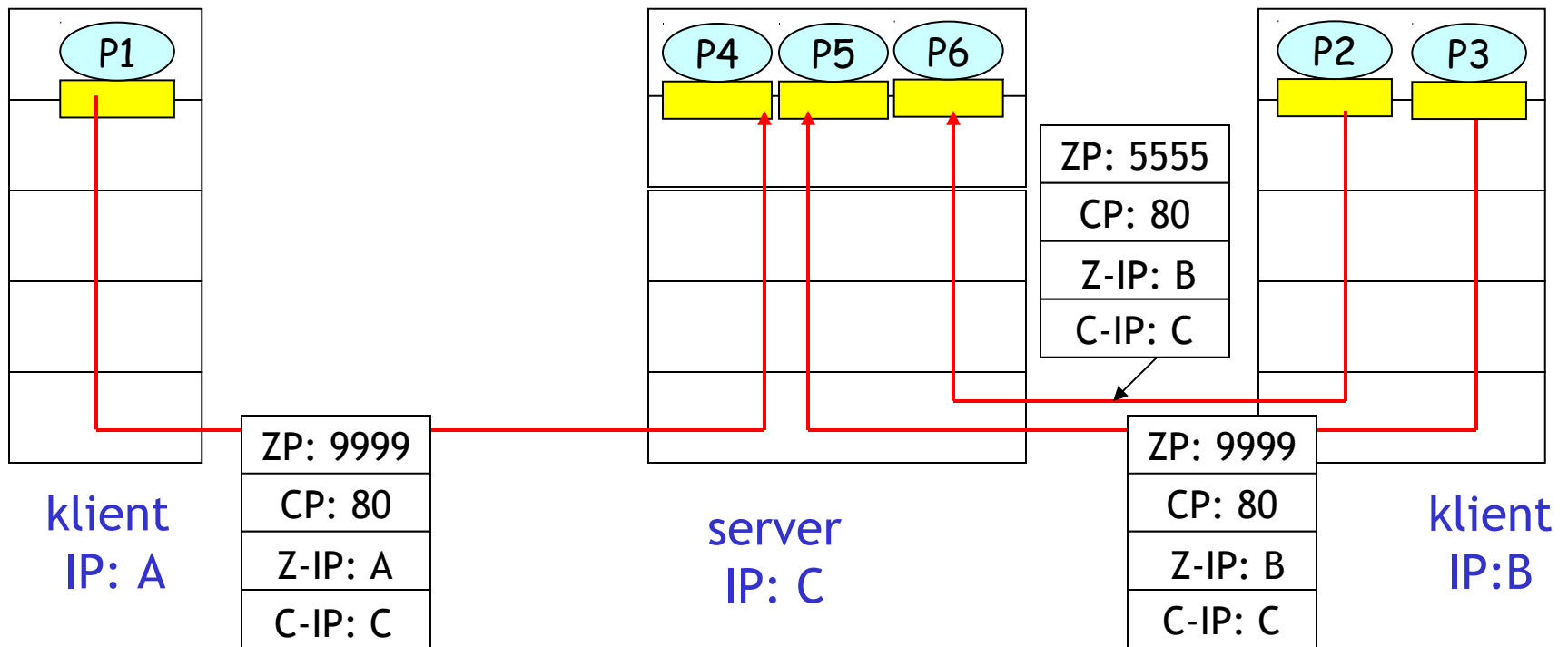
Zdrojový port poskytuje  
“návratovú adresu”

ZP = zdrojový port  
CP = cieľový port

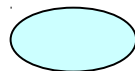
# Adresácia soketov v TCP

- ❑ TCP soket je jednoznačne identifikovaný štvoricou:
  - ❖ zdrojová IP adresa
  - ❖ zdrojový port
  - ❖ cieľová IP adresa
  - ❖ cieľový port
- ❑ príjemca použije všetky štyri hodnoty na nasmerovanie k správne mu soketu
- ❑ proces môže mať súčasne otvorených veľa TCP soketov:
  - ❖ každý soket je identifikovaný svojou štvoricou
- ❑ napr. webové servery majú pre každého napojeného klienta samostatný soket
  - ❖ všetci komunikujú s rovnakým portom

# Adresácia soketov v TCP



= soket



= proces

ZP = zdrojový port  
CP = cieľový port  
Z-IP = zdrojová IP adresa  
C-IP = cieľová IP adresa



# Osnova rozprávania o transportnej vrstve

- ❑ 3.1 Služby transportnej vrstvy
- ❑ 3.2 Delenie správ a adresácia soketov
- ❑ 3.3 UDP: bezstavový transportný protokol
- ❑ 3.4 Princípy potvrdzovaného toku dát
- ❑ 3.5 TCP: stavový transportný protokol
  - ❖ pripájanie a odpájanie
  - ❖ štruktúra segmentu
  - ❖ potvrdzovaný tok dát
  - ❖ kontrola toku dát
- ❑ 3.6 Princípy zabezpečenia kontroly zahltenia
- ❑ 3.7 Kontrola zahltenia v protokole TCP

# UDP: User Datagram Protocol [RFC 768]

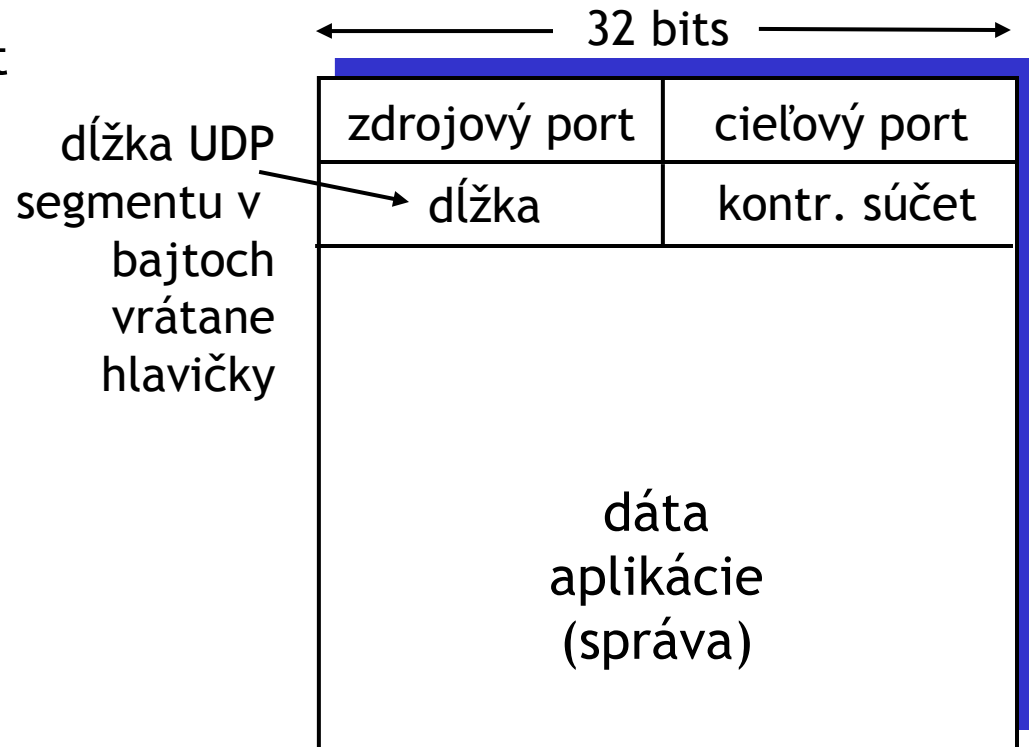
- ❑ UDP segmenty sa môžu:
  - ❖ stratit'
  - ❖ byť doručené v inom poradí, ako boli odoslané
- ❑ odosielanie „najväčším úsilím“ (best effort)
- ❑ *bezspojový protokol*:
  - ❖ žiadne nadväzovanie spojenia (handshaking) medzi odosielateľom a príjemcom
  - ❖ každý UDP segment je obslužený nezávisle na ostatných

## Načo je dobré UDP?

- ❑ žiadne inicializácie spojenia, ktoré môžu spôsobiť oneskorenie
- ❑ jednoduchý: nie je potrebné žiadne uchovávanie stavu odosielania/prijímania
- ❑ real-time prenos
- ❑ žiadna kontrola zahltenia: UDP odosiela dáta hneď, keď ich dostane z aplikačnej vrstvy
- ❑ odosielanie segmentu viacerým cieľovým staniciam (broadcast, multicast)

# UDP: použitie

- ❑ Streamovanie multimédií
  - ❖ tolerujú stratu častí dát
  - ❖ závislé na rýchlosti odosielania
  - ❖ viacerým cieľom cez multicast
- ❑ DNS
- ❑ SNMP - simple network management protocol
- ❑ spoľahlivý prenos cez UDP: spoľahlivosť je riešená v aplikačnej vrstve
  - ❖ špecifické zotavovanie z chýb pre danú aplikáciu



formát UDP segmentu

# Kontrolný súčet UDP segmentu

Ciel: odhalenie “chýb” (napr. zmenené bity) v dôjdenom segmente

## Odosielateľ:

- ❑ prechádza obsah segmentu ako množinu 16-bitových čísiel
- ❑ kontrolný súčet: sčítanie týchto čísiel a na záver vyrobenie inverzného čísla z výsledku sčítania (nuly vymení za jednotky a jednotky za nuly)
- ❑ odosielateľ pridá kontrolný súčet do UDP hlavičky

## Príjemca:

- ❑ vypočíta súčet obsahu segmentu ako množiny 16-bitových čísiel
- ❑ nakoniec sčíta výsledok s kontrolným súčtom z UDP hlavičky.
  - ❖ Ak výsledkom nie je číslo so samými jednotkami, našli sme chybu a segment zahodíme
  - ❖ Ak áno, nenašli sme chybu (ale chyba mohla aj tak nastať)

# Príklad kontrolného súčtu

- Obsah segmentu tvoria červené čísla

Odosielateľ:

	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	<hr/>															
	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
súčet	<hr/>															
	1	1	0	0	1	0	1	0	1	1	0	0	1	0	1	0
kontrolný súčet	0	0	1	1	0	1	0	1	0	0	1	1	0	1	0	1

Príjemca - spraví rovnaký súčet obsahu segmentu:

súčet	1	1	0	0	1	0	1	0	1	1	0	0	1	0	1	0
	0	0	1	1	0	1	0	1	0	0	1	1	0	1	0	1
	<hr/>															
kontrolný súčet	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

z hlavičky

same 1 - OK

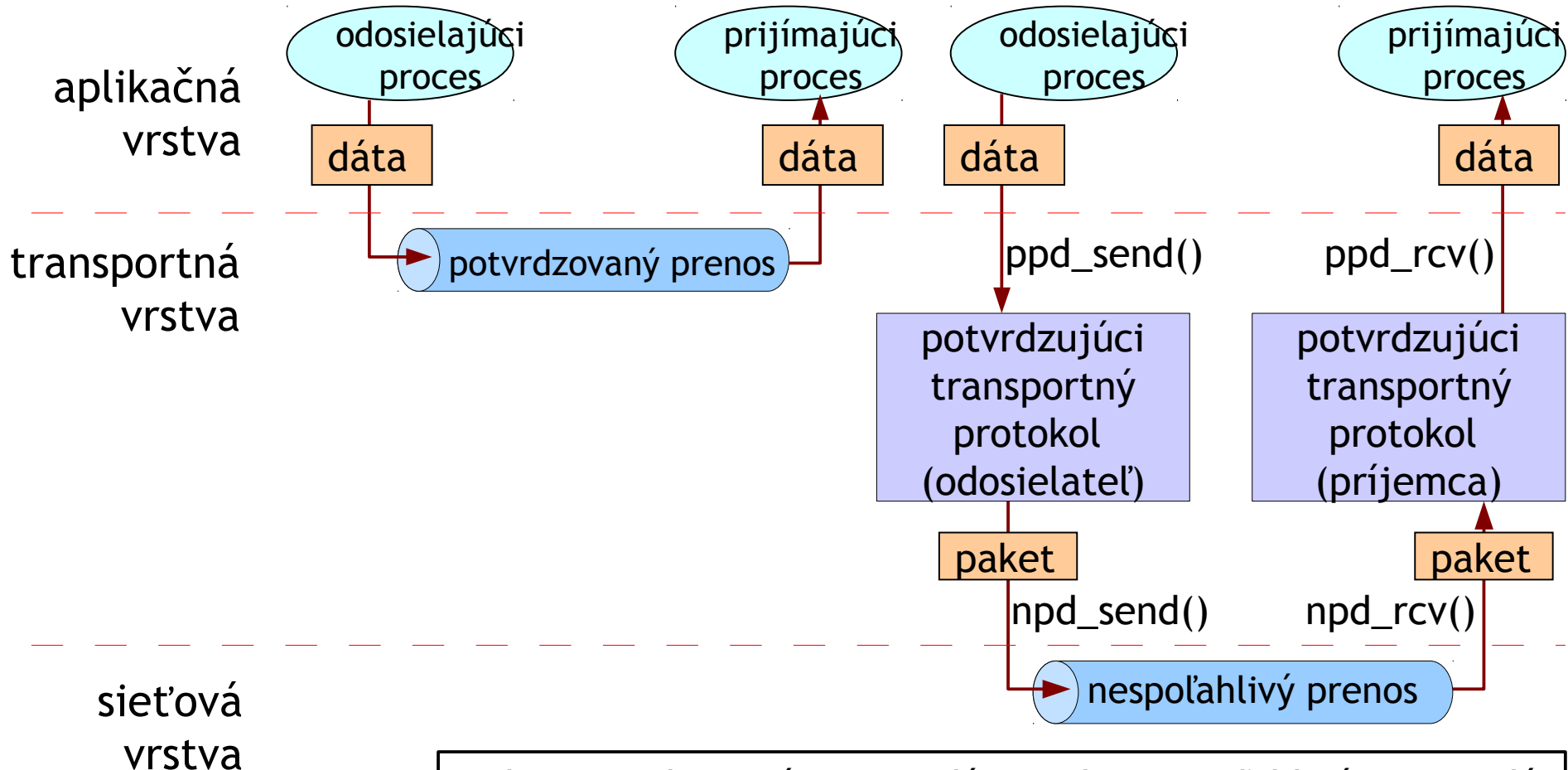
# Osnova rozprávania o transportnej vrstve

- ❑ 3.1 Služby transportnej vrstvy
- ❑ 3.2 Delenie správ a adresácia soketov
- ❑ 3.3 UDP: bezstavový transportný protokol
- ❑ 3.4 Princípy potvrdzovaného toku dát
- ❑ 3.5 TCP: stavový transportný protokol
  - ❖ pripájanie a odpájanie
  - ❖ štruktúra segmentu
  - ❖ potvrdzovaný tok dát
  - ❖ kontrola toku dát
- ❑ 3.6 Princípy zabezpečenia kontroly zahltenia
- ❑ 3.7 Kontrola zahltenia v protokole TCP

# Princípy potvrdzovaného prenosu

Poskytovaná služba:

Implementácia služby:

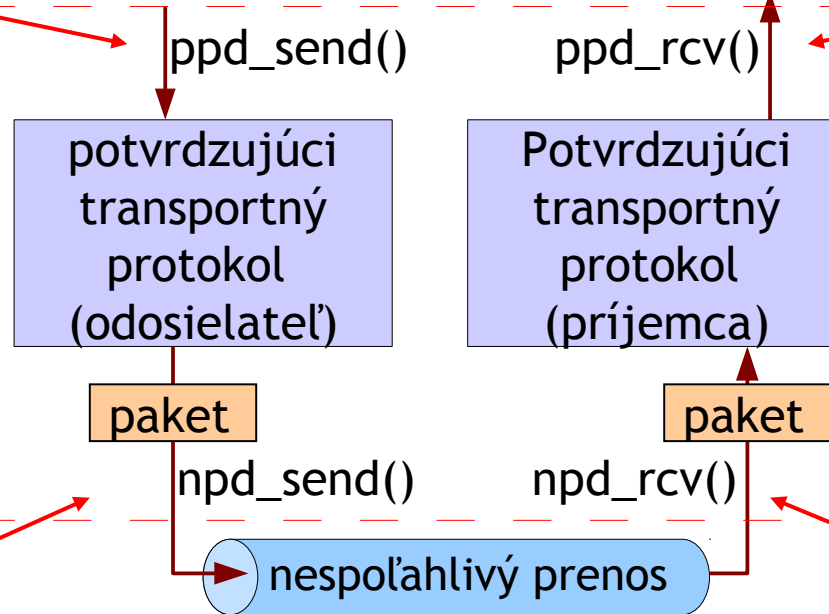


ppd = potvrdzovaný prenos dát, npd = nespoľahlivý prenos dát

# Princípy potvrdzovaného prenosu

**ppd\_send()** : volané z aplikačnej vrstvy, odovzdá dáta určené pre aplikačnú vrstvu príjemcu

**ppd\_rcv()** : zavolané potvrdzujúcim protokolom, aby odovzdal dáta aplikačnej vrstve



**npd\_send()** : volané potvrdzujúcim protokolom na odoslanie paketu príjemcovi cez nespoľahlivý kanál

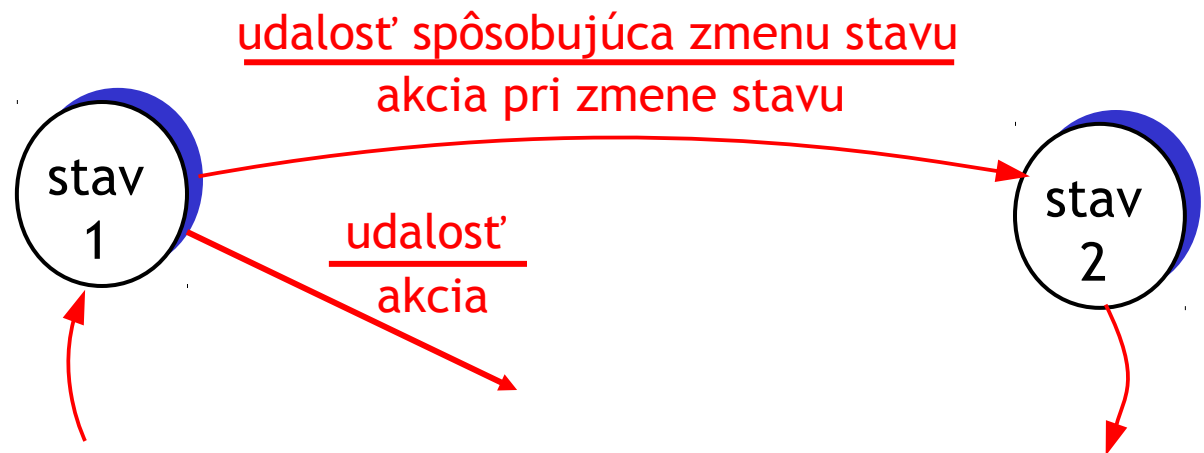
**npd\_rcv()** : zavolané, keď paket dôjde cez nespoľahlivý kanál



# Cesta k potvrdzovanému prenosu dát

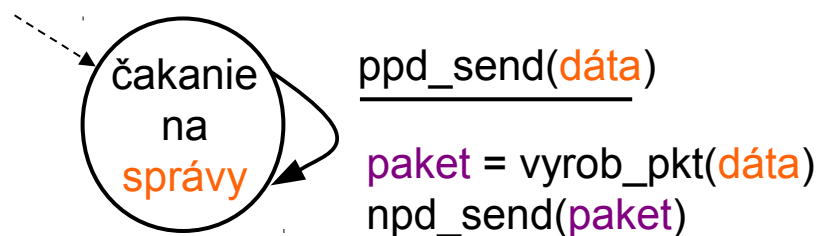
## Náš postup:

- ❑ Postupne zlepšujeme odosielateľa a príjemcu potvrdzujúceho transportného protokolu
- ❑ Uvažujeme zatiaľ iba jednosmerný prenos dát
  - ❖ iba kontrolné dáta môžu putovať oboma smermi
- ❑ Na popis odosielateľa a príjemcu použijeme konečnosťové automaty

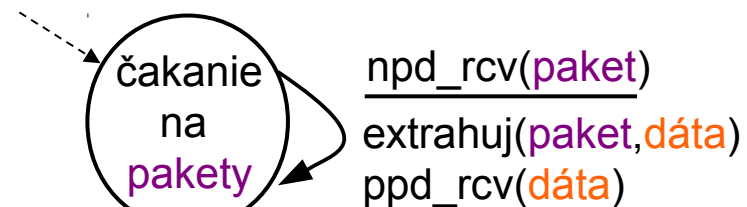


# ppd 1.0: potvrdzovaný prenos dát cez spoľahlivý kanál

- používaný kanál je dokonale spoľahlivý
  - ❖ žiadne bitové chyby
  - ❖ žiadna strata paketov
- odosielateľ aj príjemca majú jednostavový automat:
  - ❖ odosielateľ posiela dáta do kanála
  - ❖ príjemca prijíma dáta z kanála



odosielateľ



príjemca

## ppd 2.0: kanál s bitovými chybami

- ❑ používaný kanál môže prehodiť hodnotu niektorých bitov
  - ❖ použijeme kontrolný súčet na odhalenie chýb
- ❑ *otázka*: ako sa zotaviť z takýchto chýb:
  - ❖ *potvrdenia (ACKs)*: príjemca explicitne povie odosielateľovi, že paket prijal v poriadku
  - ❖ *negatívne potvrdenia (NAKs)*: príjemca explicitne povie odosielateľovi, že paket prijal s chybami
  - ❖ odosielateľ pošle paket ešte raz vždy, keď prijme NAK
- ❑ vylepšenie spd 2.0 oproti spd 1.0:
  - ❖ odhaľovanie chýb
  - ❖ spätná väzba od príjemcu: kontrolné správy (ACK,NAK)

# ppd 2.0: automaty

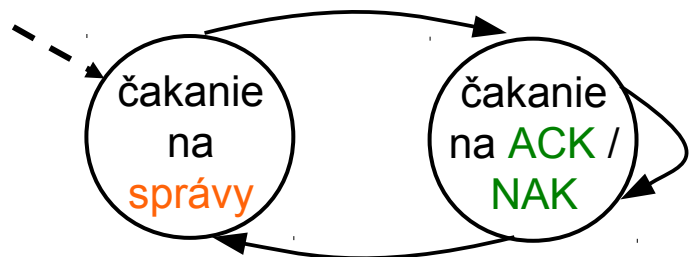
npd\_send(dáta)

paket = make\_pkt(dáta, kontr. súčet)

npd\_send(paket)

npd\_rcv(ppaket) &&  
isNAK(ppaket)

npd\_send(paket)



npd\_rcv(ppaket) && isACK(ppaket)

(nič)

odosielateľ

príjemca

npd\_rcv(paket) &&  
chybný(paket)

npd\_send(NAK)



npd\_rcv(paket) &&  
bezchybný(paket)

extrahuj(paket, dáta)

ppd\_rcv(dáta)

npd\_send(ACK)

# ppd 2.0 má fatálny nedostatok!

## Čo ak sú ACK/NAK poškodené?

- ❑ odosielateľ nevie čo sa stalo na strane príjemcu!
- ❑ nemôže jednoducho preposlať: možná duplicita

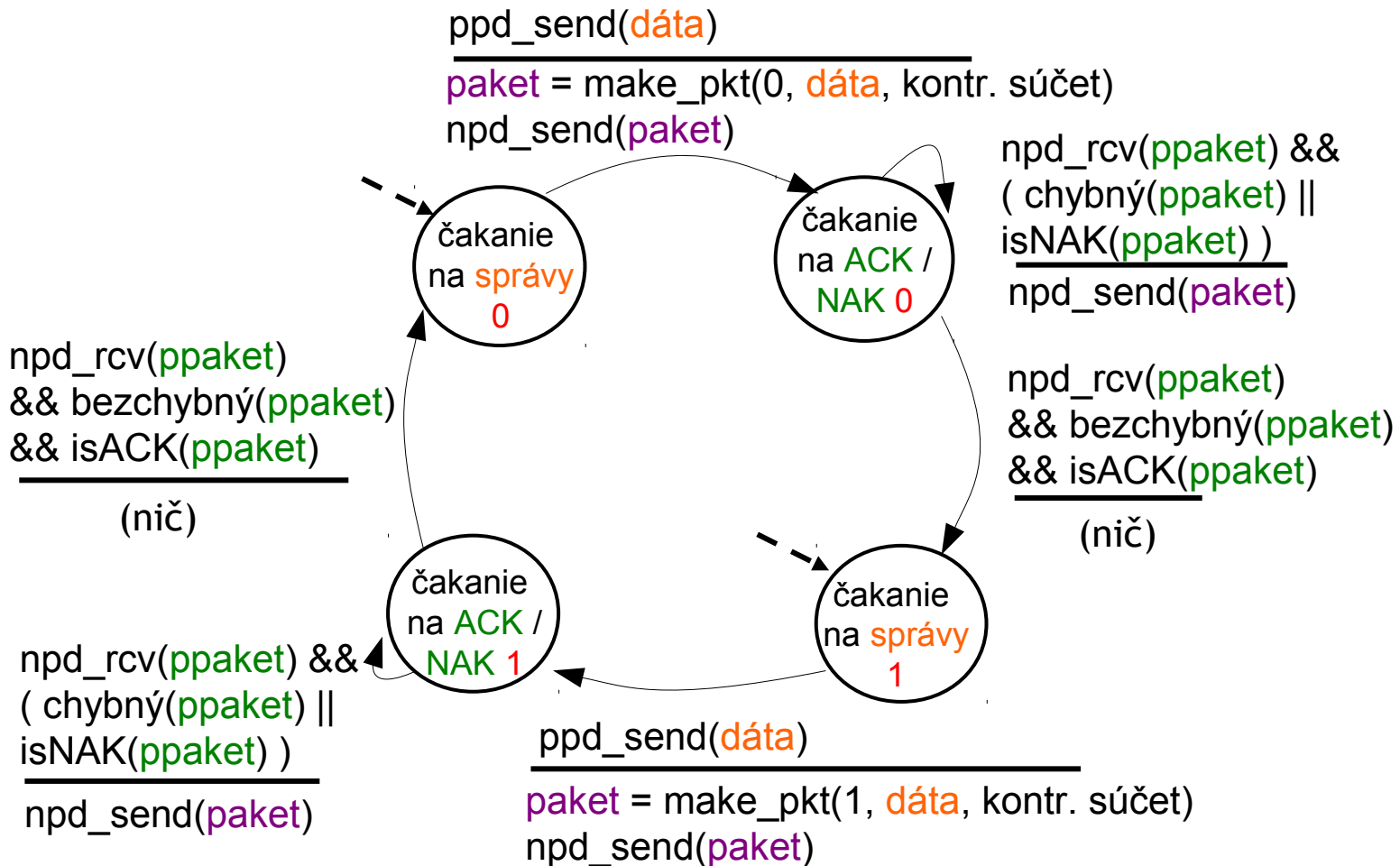
## Riešenie duplicit:

- ❑ odosielateľ znova pošle aktuálny paket, ak je ACK/NAK poškodené
- ❑ odosielateľ pridá *sekvenčné číslo* každému paketu
- ❑ príjemca duplicity zahadzuje

### — stop and wait —

Odosielateľ čaká po každom odoslanom pakete na odpoveď príjemcu

# ppd 2.1: odosielateľ kt. zvládne poškodené ACK/NAK



# ppd 2.1: príjemca

npd\_rcv(paket) && bezchybný(paket)  
&& has\_seq0(paket)

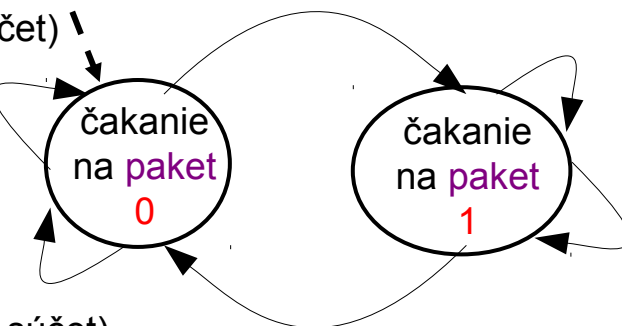
extrahuj(paket,dáta)  
ppd\_rcv(dáta)  
ppaket = make\_pkt(ACK, kontr. súčet)  
npd\_send(ppaket)

npd\_rcv(paket) && chybný(paket)

ppaket = make\_pkt(NAK, kontr. súčet)  
npd\_send(ppaket)

npd\_rcv(paket) &&  
bezchybný(paket) &&  
has\_seq1(paket)

ppaket = make\_pkt(ACK, kontr. súčet)  
npd\_send(ppaket)



npd\_rcv(paket) && bezchybný(paket)  
&& has\_seq1(paket)

extrahuj(paket,dáta)  
ppd\_rcv(dáta)  
ppaket = make\_pkt(ACK, kontr. súčet)  
npd\_send(ppaket)

npd\_rcv(paket) && chybný(paket)

ppaket = make\_pkt(NAK, kontr. súčet)  
npd\_send(ppaket)

npd\_rcv(paket) &&  
bezchybný(paket) &&  
has\_seq0(paket)

ppaket = make\_pkt(ACK, kontr. súčet)  
npd\_send(ppaket)

# ppd 2.1: diskusia

## Odosielateľ:

- ❑ pridanie sekvenčného čísla do paketu
- ❑ dve sekvenčné čísla (0,1) stačia. Prečo?
- ❑ musí zisťovať chybovosť prijatých ACK/NAK paketov
- ❑ dvojnásobok stavov
  - ❖ extra stavy si “pamätajú”, či odoslaný paket mal číslo 0 alebo 1

## Príjemca:

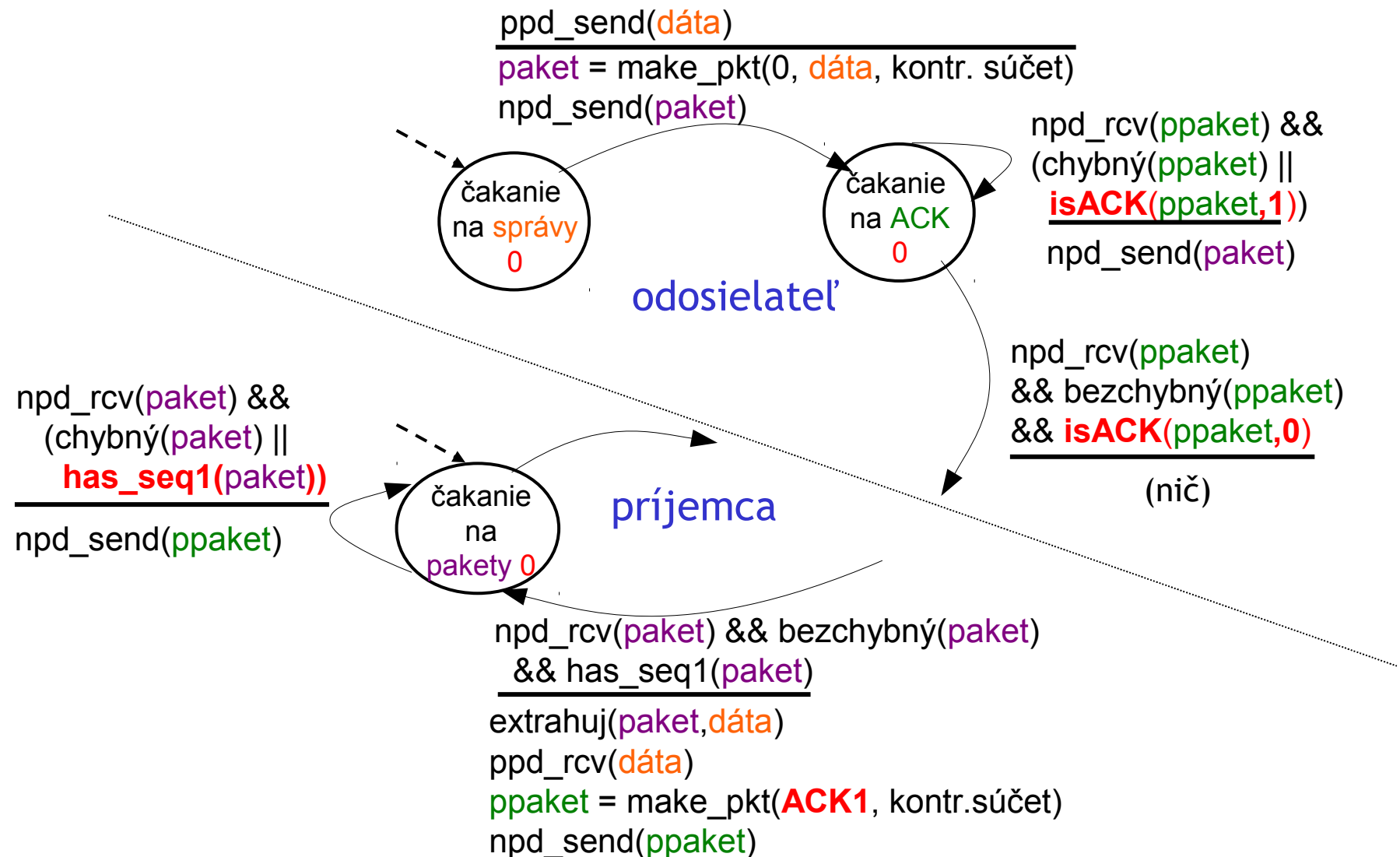
- ❑ musí zisťovať duplicitu paketov
  - ❖ stav identifikuje, či očakávame paket so sekvenčným číslom 0 alebo 1
- ❑ poznámka: príjemca nevie, či ACK/NAK bol odosielateľovi doručený bez chyby



## ppd 2.2: protokol bez NAK správ

- ❑ rovnaká funkcionálnosť ako ppd 2.1, ale používame už iba ACK potvrdenia
- ❑ namiesto správy NAK, príjemca odošle ACK so sekvenčným číslom posledného bezchybného paketu
  - ❖ potvrdzovací ACK paket musí obsahovať číslo paketu, ktorý potvrdzujeme
- ❑ dva ACK pakety s rovnakým číslom majú rovnaký dôsledok ako NAK: *pošli aktuálny paket znova*

# ppd 2.2: polovice automatov odosielaťa a príjemcu



# ppd 3.0: kanál s bitovými chybami a stratami paketov

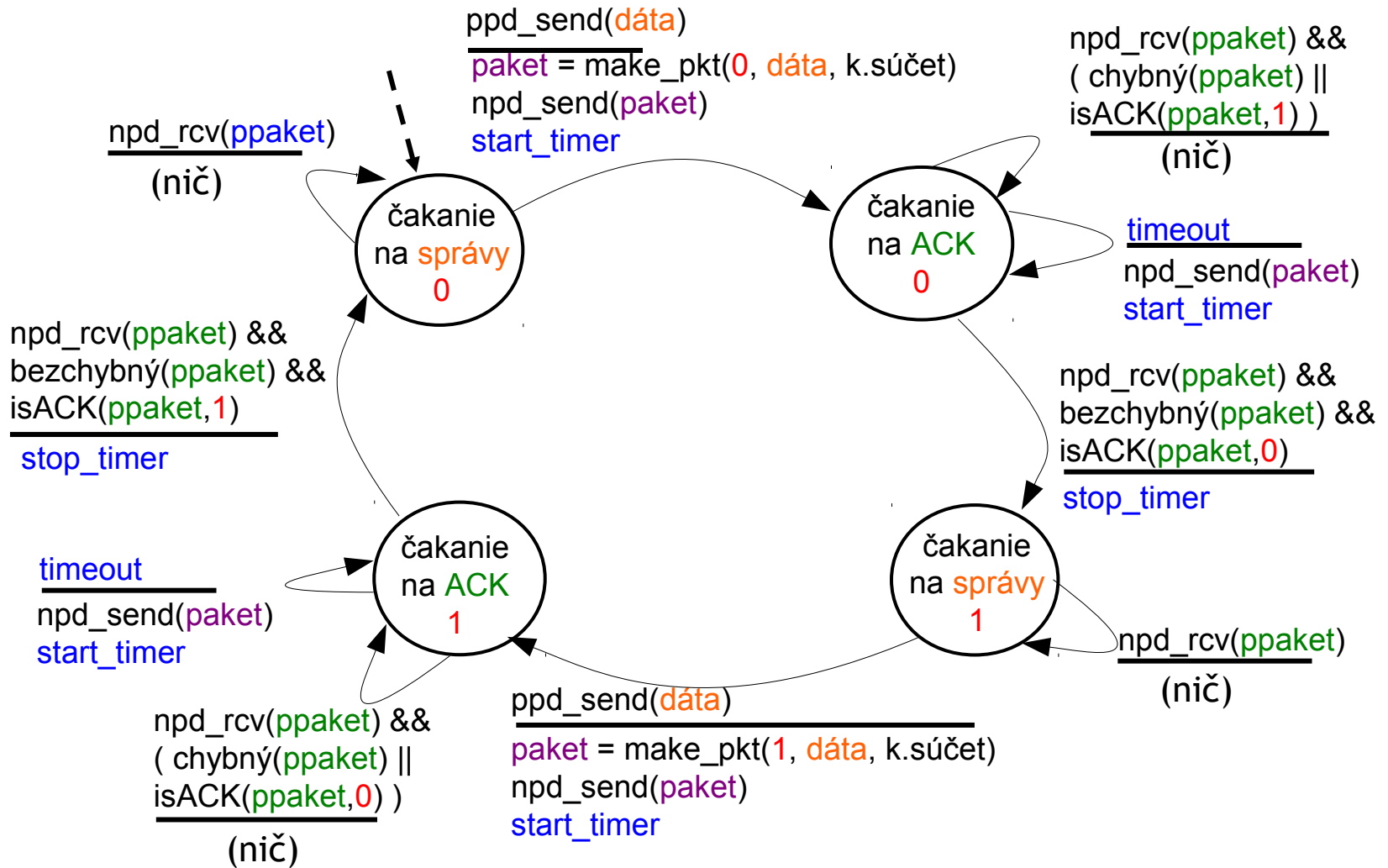
Nový predpoklad: použitý kanál môže aj stratit' pakety (dáta alebo potvrdenia)

- ❖ kontrolný súčet, sekvenčné číslo a potvrdenia ACK využijeme, ale to nestačí

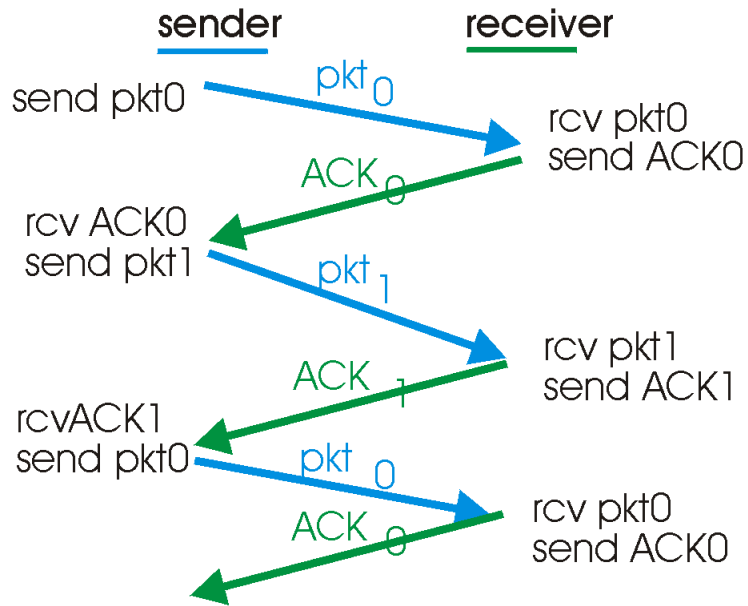
Prístup: odosielateľ čaká “rozumný” čas na ACK

- ❑ ak v tomto čase žiadne ACK neprišlo, pošle paket znova
- ❑ ak sa paket (alebo ACK) iba oneskorí (nestratí):
  - ❖ posielanie bude spôsobovať duplicity, ale to vieme vyriešiť cez sekvenčné číslo
  - ❖ príjemca musí špecifikovať sekvenčné číslo v ACK
- ❑ vyžaduje “časovač”: meranie času od posledného odoslania paketu

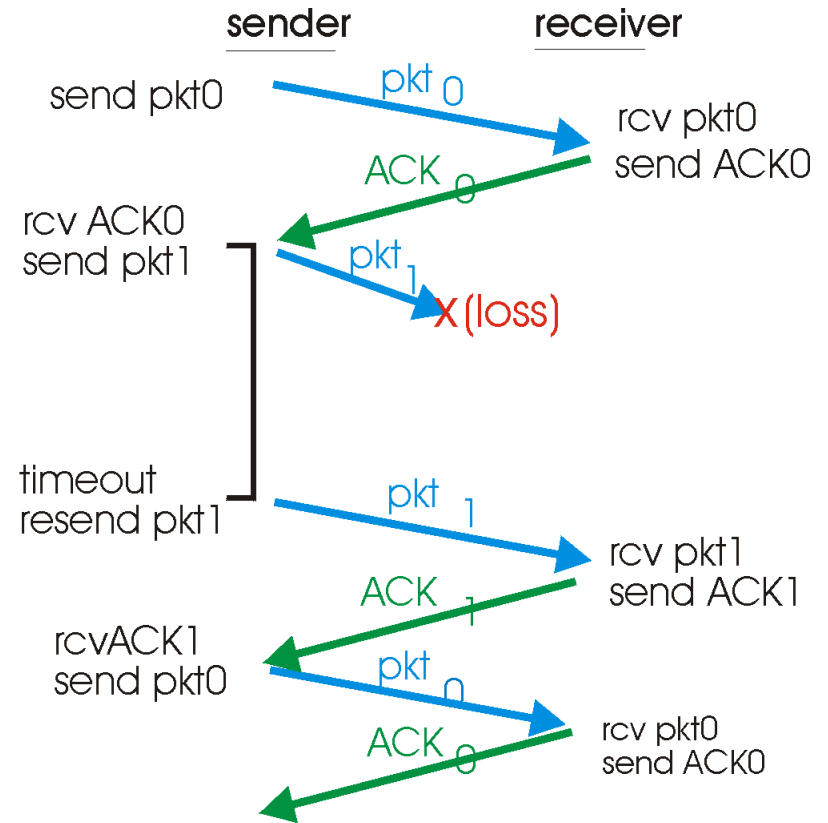
# ppd 3.0 odosielateľ



# ppd 3.0 v čase

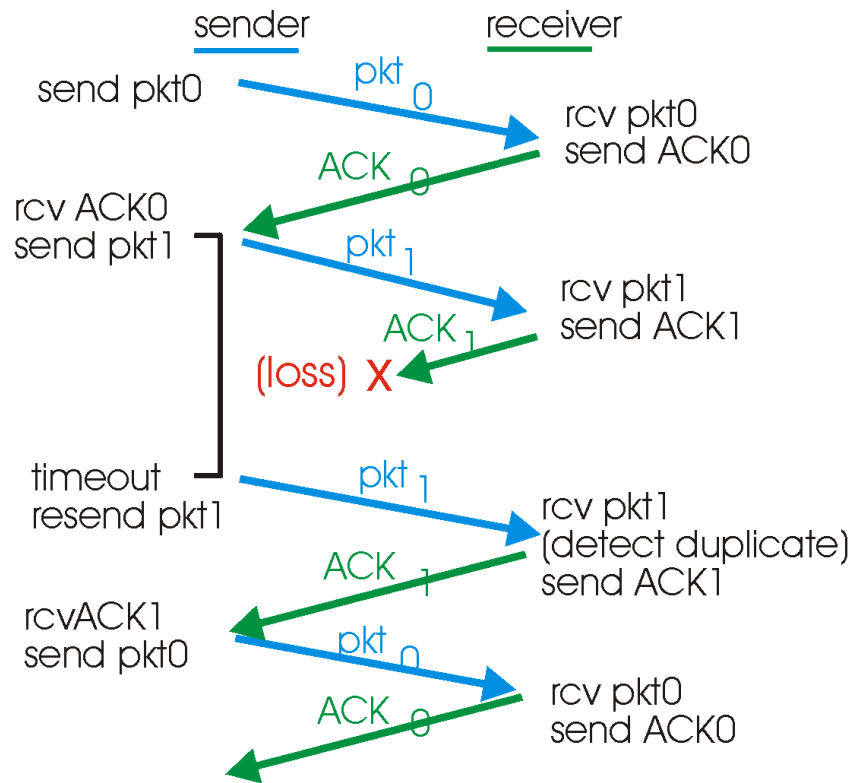


(a) operation with no loss

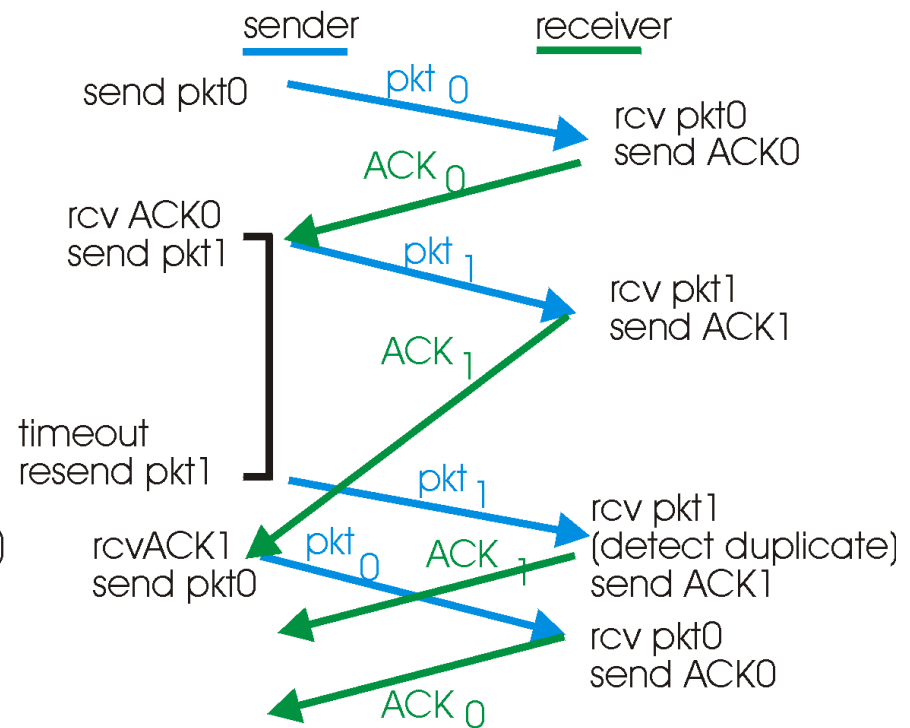


(b) lost packet

# ppd 3.0 v čase



(c) lost ACK



(d) premature timeout

# rýchlosť ppd 3.0

- ❑ ppd 3.0 funguje (použitie je napr. vo WiFi) , ale výkon je biedny
- ❑ príklad: 1 Gb/s spoj, zdržanie prenosom 15 ms, pakety veľkosti 1kB:

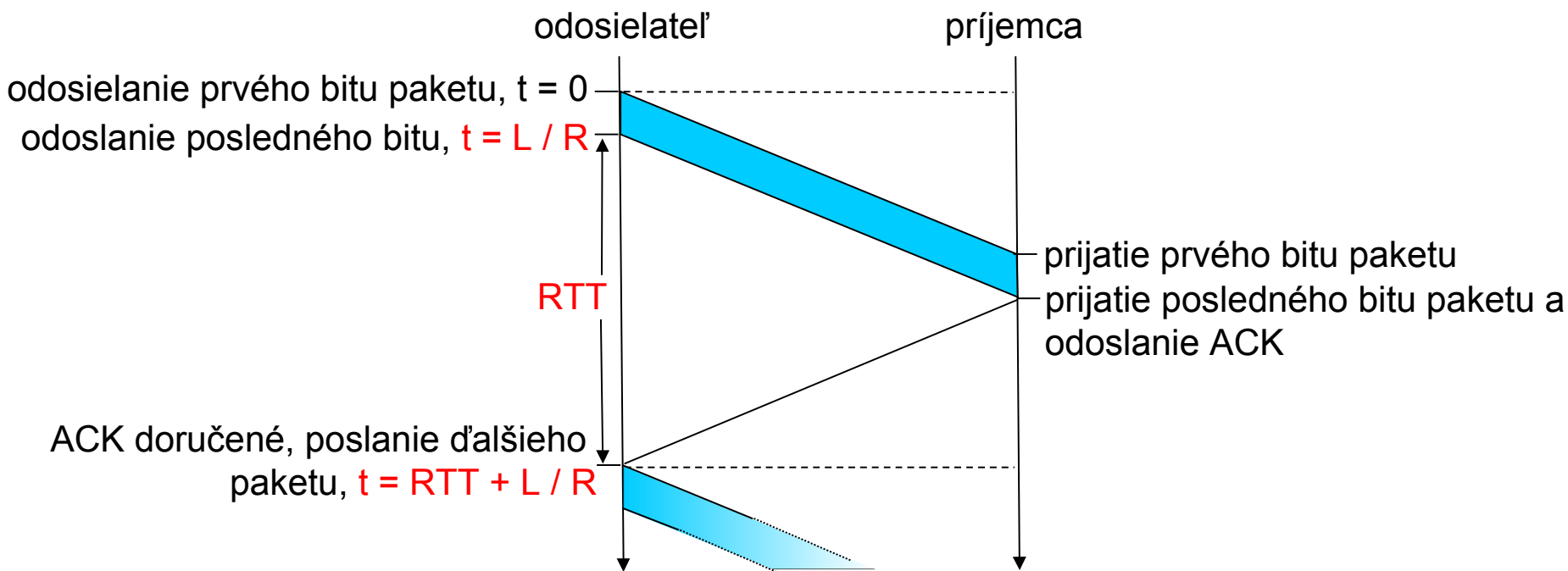
$$T_{\text{odoslanie}} = \frac{L \text{ (počet bitov paketu)}}{R \text{ (šírka pásma v b/s)}} = \frac{8\text{kb}}{10^9 \text{ b/s}} = 8 \text{ mikrosekúnd}$$

- ❖ E: **efektivita** - zlomok času v ktorom sa odosiela

$$E = \frac{L / R}{RTT + L / R} = \frac{0,008 \text{ ms}}{30,008 \text{ ms}} = 0.00027$$

- ❖ 1kB paket za 30 ms -> 33kB/s prenos cez 1 Gb/s spoj
- ❖ protokol obmedzuje fyzické možnosti spoja!

# ppd3.0: stop-and-wait



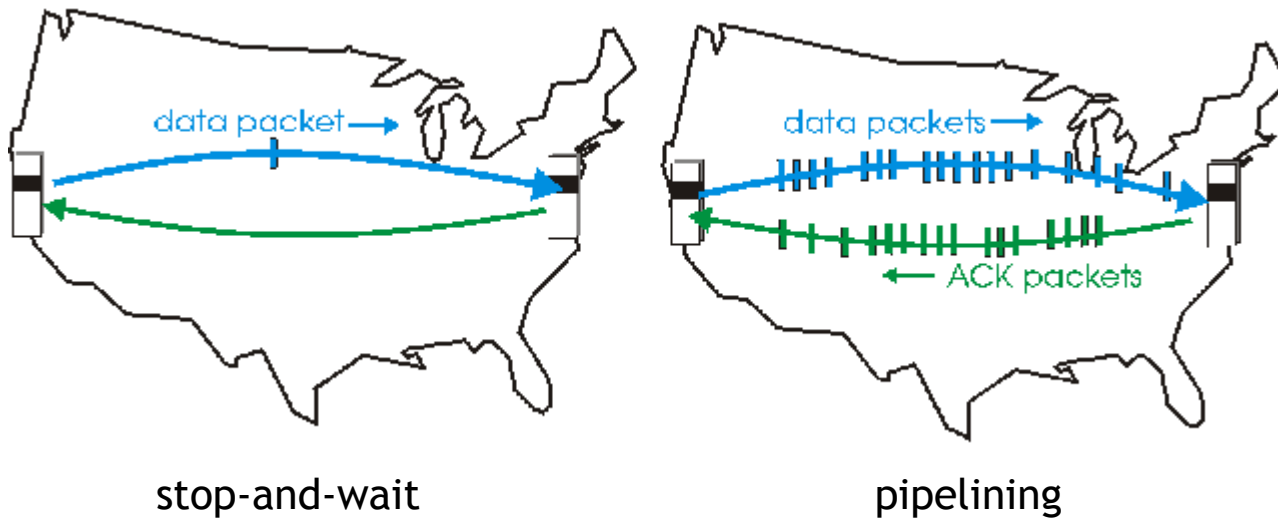
$$E = \frac{L / R}{RTT + L / R} = \frac{0,008 \text{ ms}}{30,008 \text{ ms}} = 0.00027$$



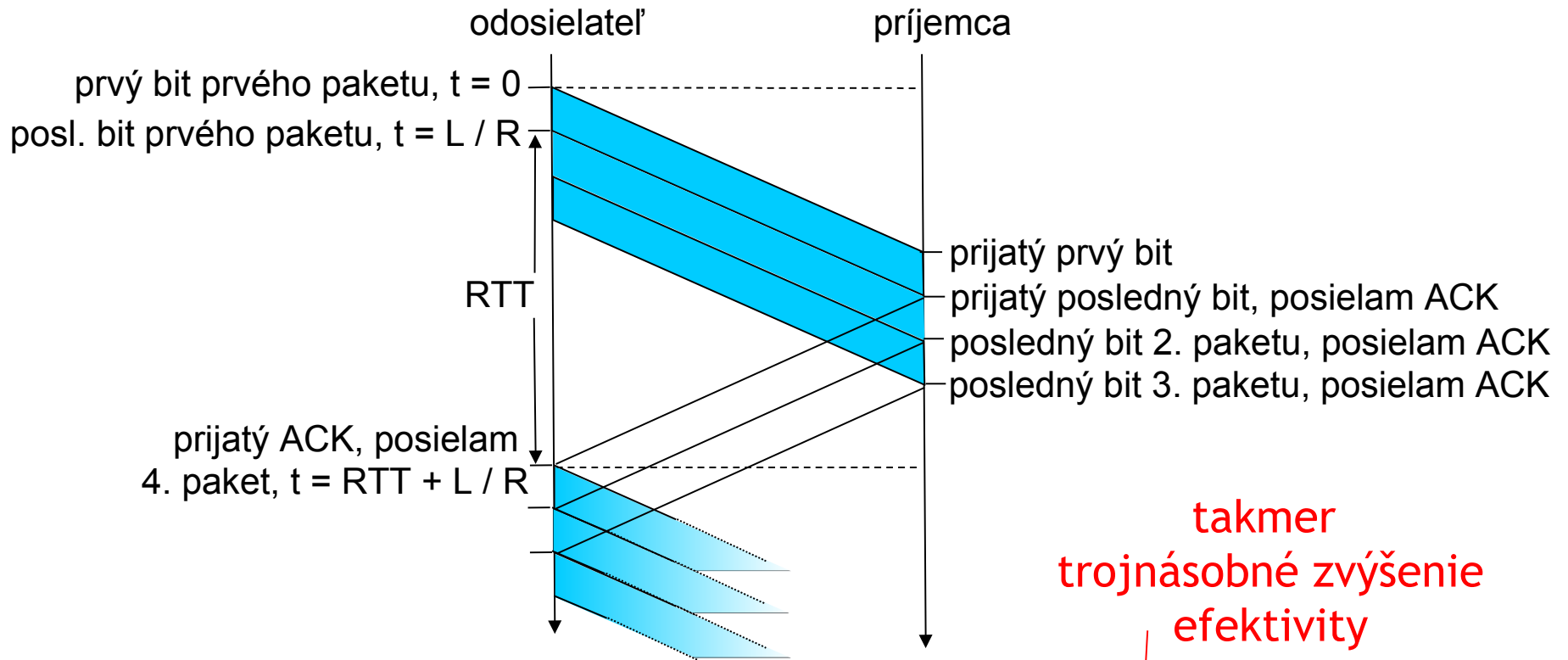
# Protokoly s pipeliningom

**Pipelining:** odosielateľ odosiela viac paketov, pokiaľ mu prídu ACK pakety

- ❖ rozsah sekvenčných čísiel sa musí zväčšiť
- ❖ odosielateľ aj príjemca musia udržiavať buffer paketov



# Pipelining: zvýšenie efektivity

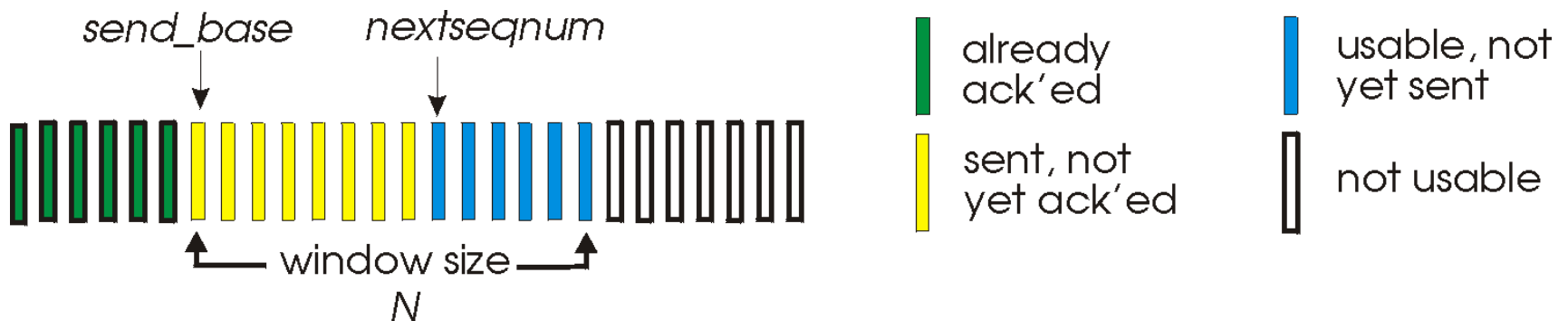


$$E = \frac{3 * L / R}{RTT + 3 * L / R} = \frac{0,024 \text{ ms}}{30,024 \text{ ms}} = 0.0008$$

# Go-Back-N

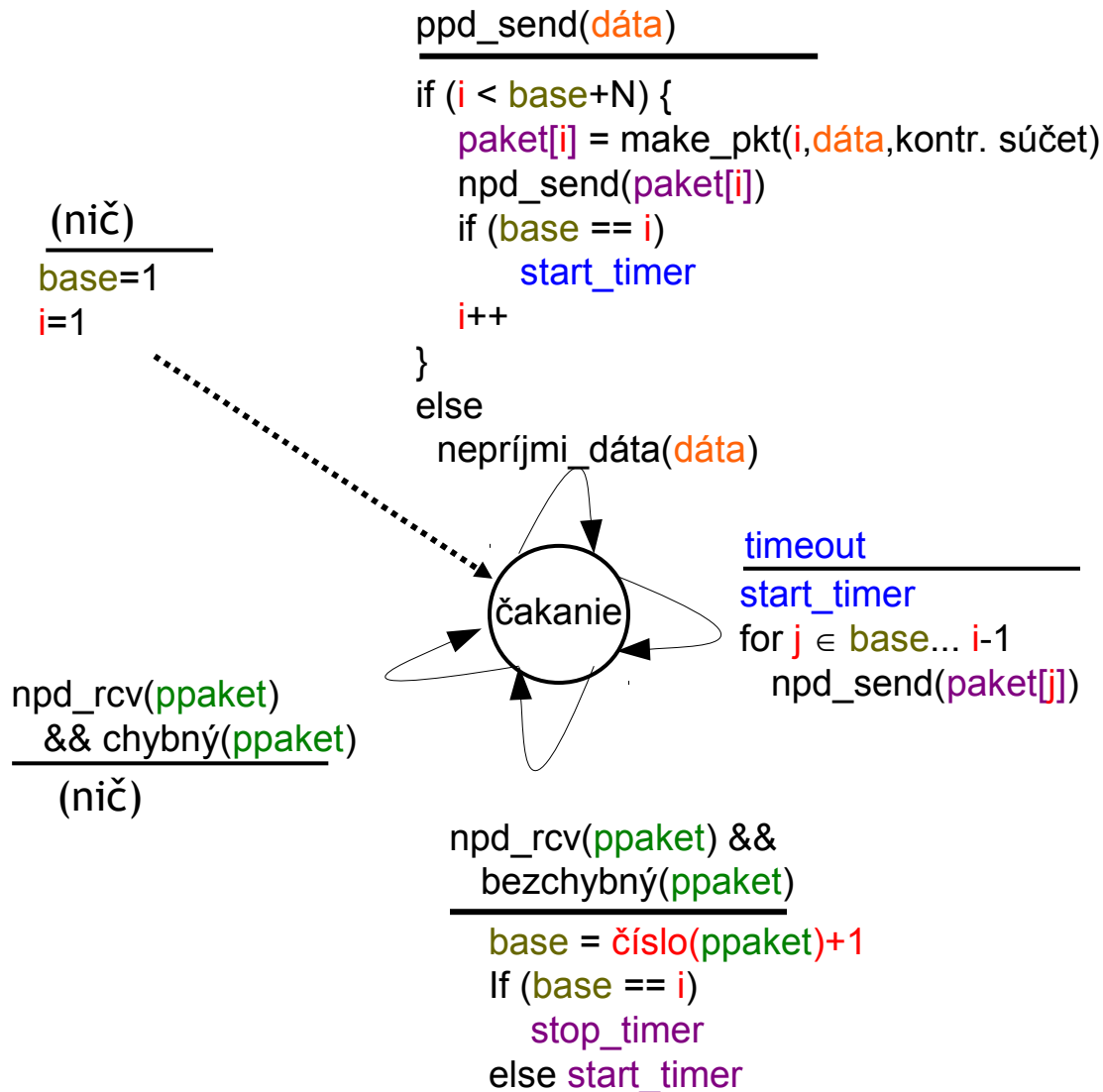
## Odosielateľ:

- ❑ k-bitové sekvenčné číslo v hlavičke paketu
- ❑ “okno” veľkosti  $N$  určujúce, koľko odoslaných paketov nemusí byť potvrdených

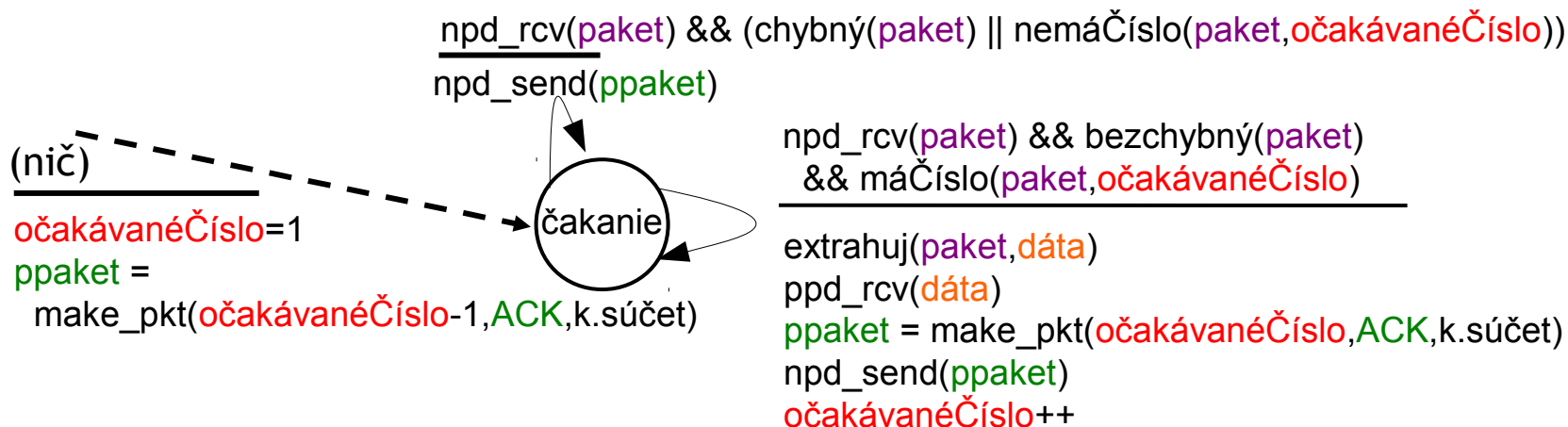


- ❑ ACK( $n$ ): kumulatívne potvrdenie za prijatie súvislej časti paketov až po sekvenčné číslo  $n$  vrátane - “kumulatívny ACK”
  - ❖ aj tu sa môže stať, že prídu rovnaké ACK pakety
- ❑ časovač pre okno paketov (štart časovača pri novom *send\_base*)
- ❑ timeout: vykonáme opätovné preposlanie paketu na *send\_base* pozíciu a aj všetkých odoslaných paketov s vyšším číslom

# Go-Back-N: odosielateľ

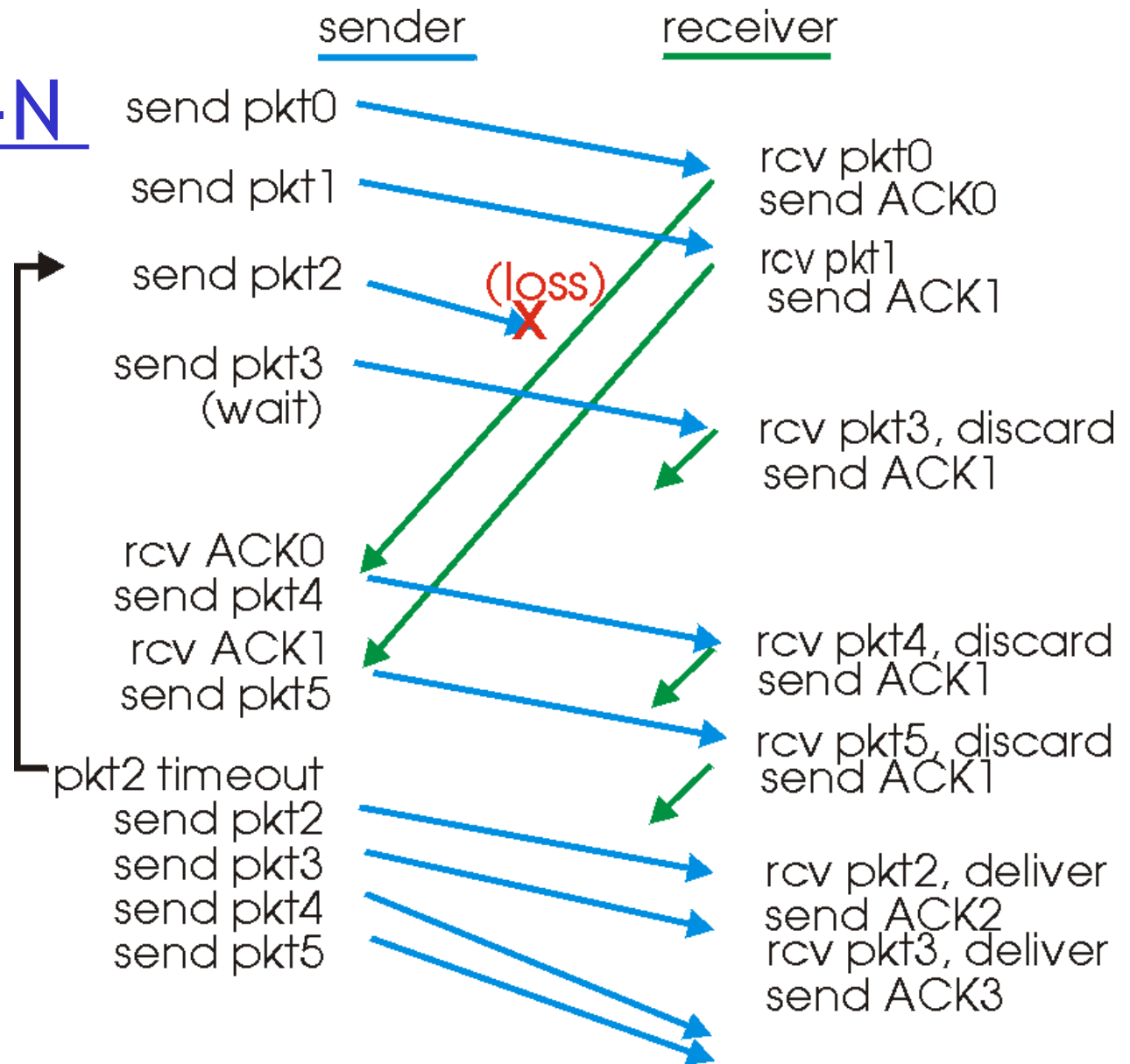


# Go-Back-N: príjemca



- vždy odošle ACK pre bezchybne doručený paket s najväčším číslom v poradí bezchybne doručených
  - ❖ môžeme posielat' rovnaké ACK viac krát
  - ❖ potrebujeme si pamätať iba očakávané číslo
- pakety mimo poradia doručených paketov:
  - ❖ zahodíme (nejdú do buffra) -> **buffer príjemca nepotrebuje!**
  - ❖ opäť pošleme ACK s posledným číslom v poradí správne doručených

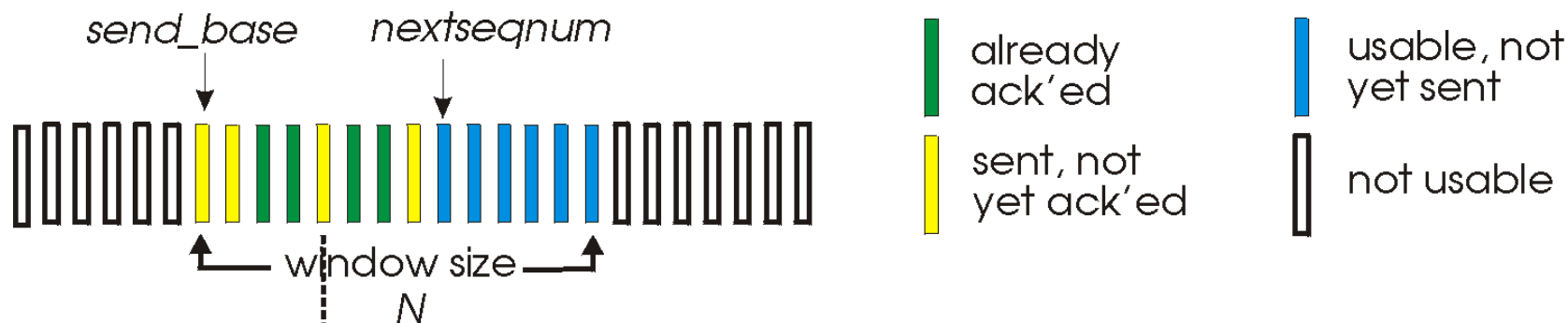
# Go-Back-N



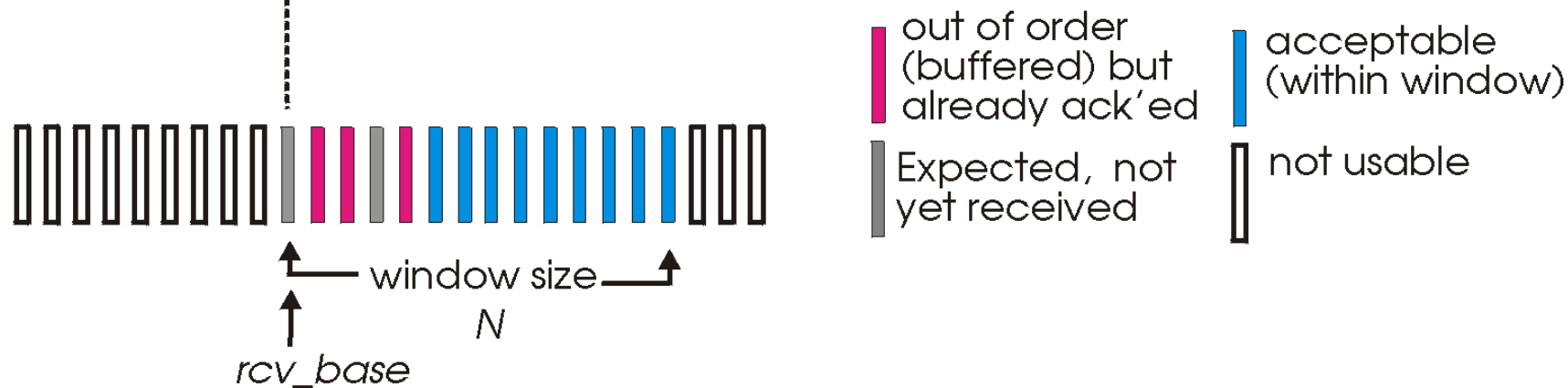
# Selektívne opakovanie

- príjemca potvrdzuje každý bezchybný paket zvlášť
  - ❖ buffer pre pakety, ktoré majú vyššie čísla ako bezchybne doručený paket s najväčším číslom v poradí bezchybne doručených
- odosielateľ preposiela znova iba tie pakety, pre ktoré neprijal potvrdzovacie ACK pakety
  - ❖ časovač pre každý nepotvrdený paket
  - ❖ okno odoslaných paketov a povolených na odoslanie

# Selektívne opakovanie: okná odosielateľa aj príjemcu



(a) sender view of sequence numbers



(b) receiver view of sequence numbers



# Selektívne opakovanie

## odosielateľ

### dáta z aplikačnej vrstvy:

- ❑ ak je voľné sekvenčné číslo v okne, pošle paket

### timeout(n):

- ❑ pošle znova paket s číslom n a reštartuje jeho časovač

### ACK(n) v okne odosielača:

- ❑ označí paket n ako doručený
- ❑ ak n bol paket s najmenším číslom v okne, posuň okno tak, aby začínalo na prvom nepotvrdenom pakete

## príjemca

### paket s číslom n z intervalu [rcv\_base, rcv\_base+N-1]

- ❑ pošli ACK(n)
- ❑ mimo poradia: daj do buffra
- ❑ v poradí: pošli paket aplik. vrstve spolu so všetkými z buffra v poradí za ním, a posuň okno k ďalšiemu nedoručenému paketu

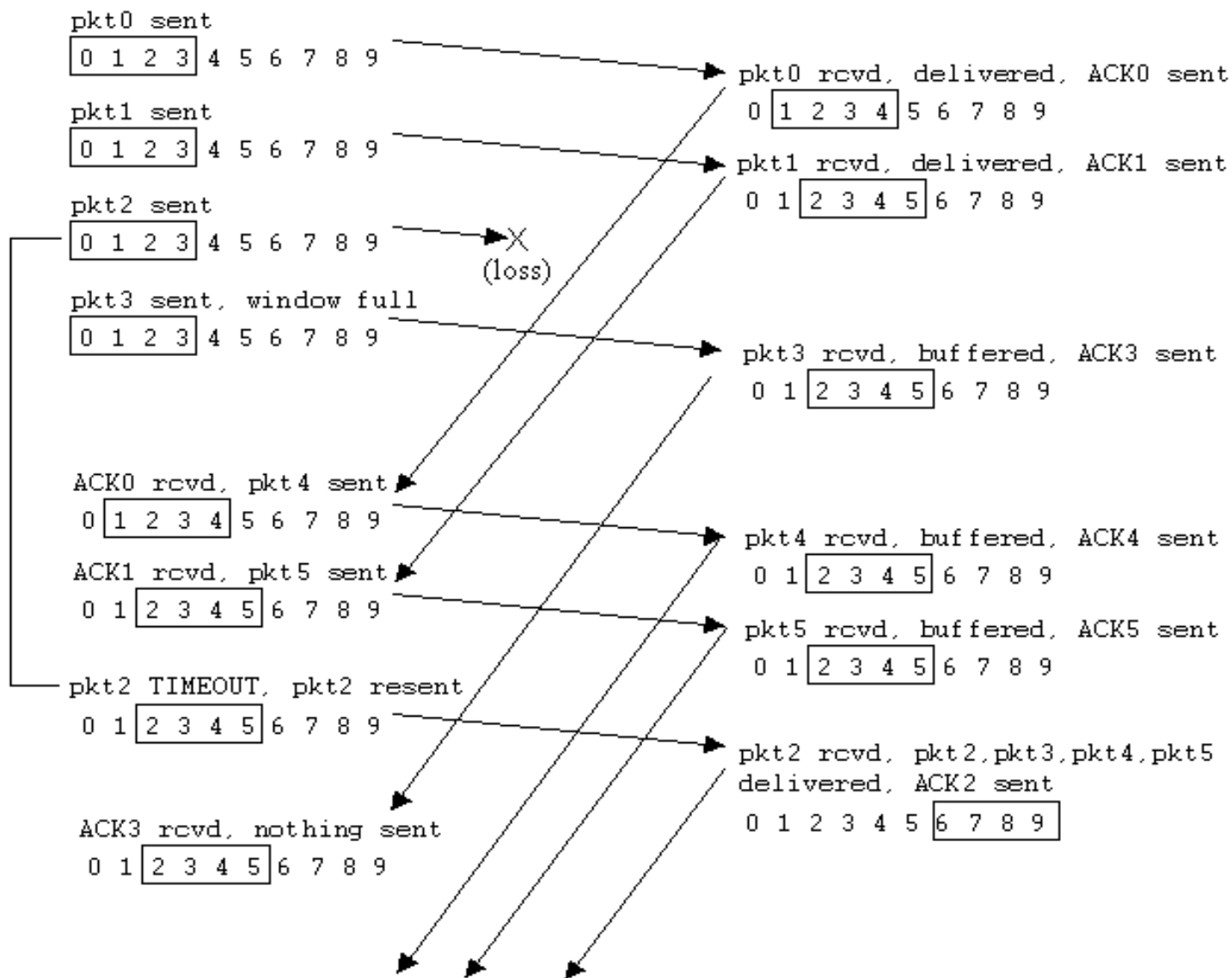
### paket s číslom n z intervalu [rcvbase-N,rcvbase-1]

- ❑ pošli ACK(n)

### inak:

- ❑ ignoruj

# Selektívne opakovanie v čase



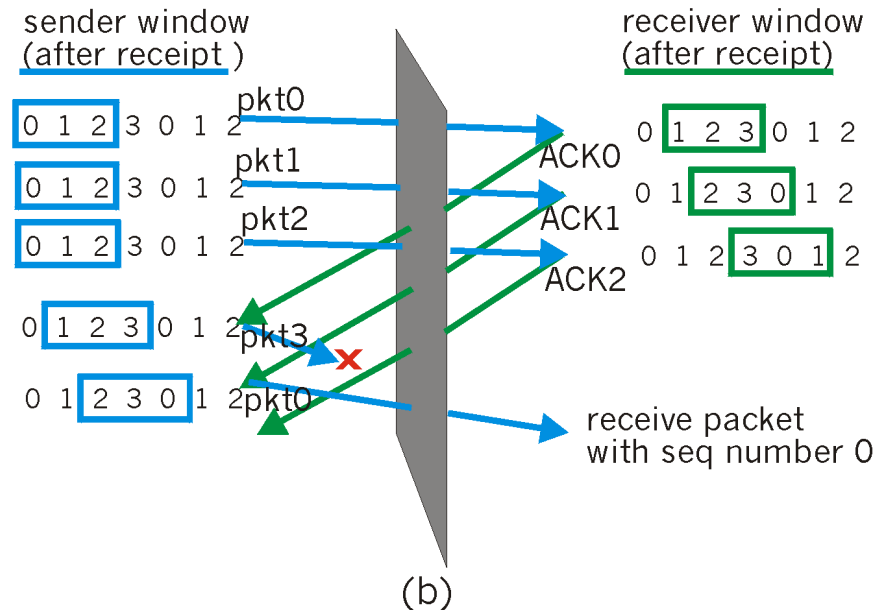
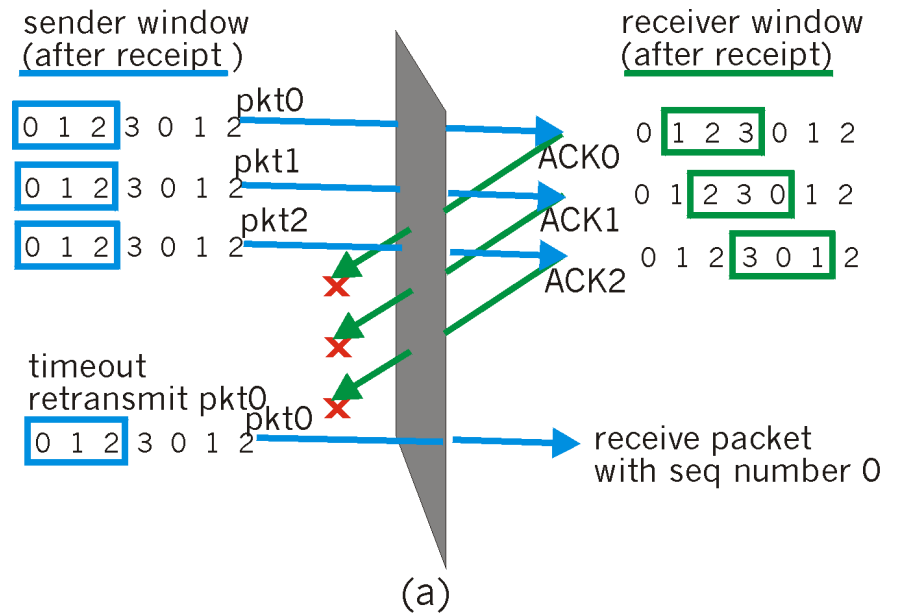
# Selektívne opakovanie

Príklad:

- sekv. čísla: 0, 1, 2, 3
- veľkosť okna=3

- príjemca vidí v oboch prípadoch to isté!
- nesprávne opätovné posielanie, ktoré sa javí ako nové

**Otázka:** aký je vzťah medzi počtom sekvenčných čísiel a veľkosťou okna?



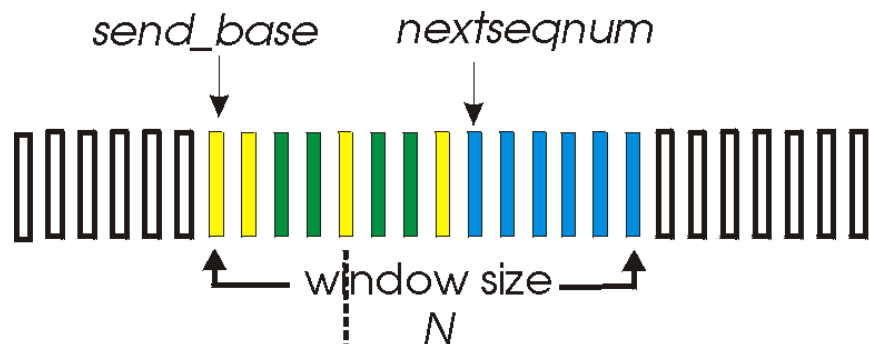
# Osnova rozprávania o transportnej vrstve

- ❑ 3.1 Služby transportnej vrstvy
- ❑ 3.2 Delenie správ a adresácia soketov
- ❑ 3.3 UDP: bezstavový transportný protokol
- ❑ 3.4 Princípy potvrdzovaného toku dát
- ❑ 3.5 TCP: stavový transportný protokol
  - ❖ pripájanie a odpájanie
  - ❖ štruktúra segmentu
  - ❖ potvrdzovaný tok dát
  - ❖ kontrola toku dát
- ❑ 3.6 Princípy zabezpečenia kontroly zahltenia
- ❑ 3.7 Kontrola zahltenia v protokole TCP

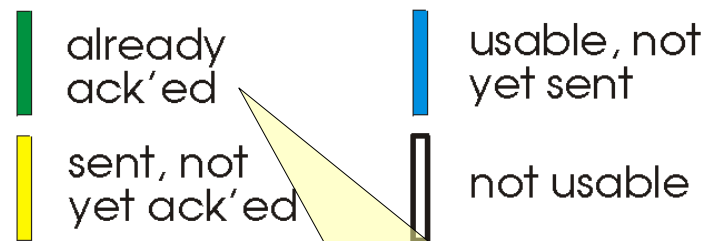
# Potvrdzovaný prenos dát cez TCP

- ❑ TCP využíva nespoľahlivý (best effort) prenos dát protokolom IP
- ❑ pipelining segmentov
  - ❖ okno príjemcu aj odosielateľa
- ❑ kumulatívne potvrdenia
- ❑ jediný časovač na preposielania
- ❑ opätovné posielanie je spôsobené:
  - ❖ timeout-om
  - ❖ viacnásobným potvrdením
- ❑ na začiatok uvažujme zjednodušeného odosielateľa:
  - ❖ ignorujme kontrolu toku dát a zahltenia

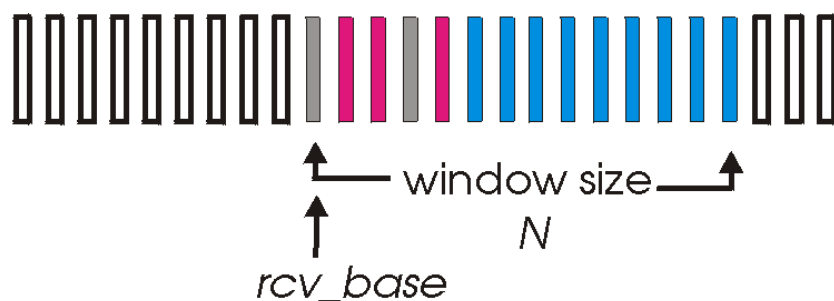
# Okná v TCP: takmer ako pri selektívnom opakovaní



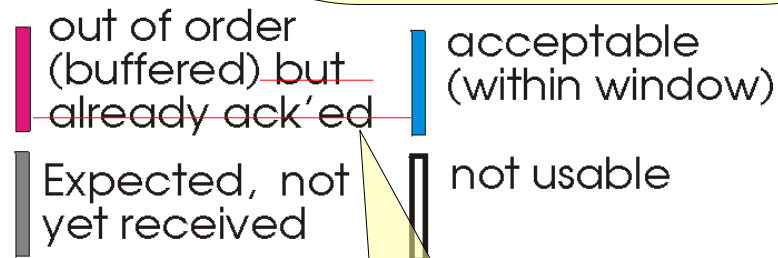
(a) sender view of sequence numbers



takéto okamžite spôsobia posunutie okna (kumulatívne potvrdenie)



(b) receiver view of sequence numbers



potvrďujeme vždy číslom rcv\_base

# Ďakujem za pozornosť

Modifikované slajdy z knihy:

*Computer Networking: A Top Down Approach* ,  
4<sup>th</sup> edition.

Jim Kurose, Keith Ross  
Addison-Wesley, July 2007.