

# Lekce 6: Techniky přenosu dat

*Jiří Peterka*

# co jsou "techniky přenosu dat"?

- **obecně:**

- všechno, co se týká přenosu celých bloků dat
  - metody, postupy, ....
  - ve smyslu:
    - když už máme zajištěn přenos jednotlivých bitů a přenášíme data po větších kvantech (blocích), než jsou jednotlivé bity

- **patří sem:**

- přenos dat v sítích, fungujících na principu přepojování okruhů
  - tzv. sériové komunikace
- přenos dat v sítích, fungujících na principu přepojování paketů
  - paketový přenos
    - přenos paketů, rámců, buněk, ...
  - spolehlivý a nespolehlivý přenos
  - spojovaný a nespojovaný přenos
  - přenos best effort vs. podpora QoS

- **dále také (mimo jiné):**

- framing („rámování“)
  - jak správně rozpoznávat celé bloky dat (rámce, buňky, pakety ...)
    - jak v proudu přijímaných bitů, bytů či znaků rozpoznat začátek a konec bloku
- **zajištění transparency dat**
  - aneb: jak poznat, kdy přenášená data jsou
    - příkazy, které je třeba interpretovat
    - „čistá data“, která je třeba přenést a nijak neměnit
- **zajištění spolehlivosti přenosu**
  - detekce chyb
  - potvrzování
    - eventuální opakování přenosu
- **řízení toku**
  - aby odesílatel nezahltl příjemce
- **předcházení zahlcení**
  - aby přenos nezahltl přenosovou síť
- .....

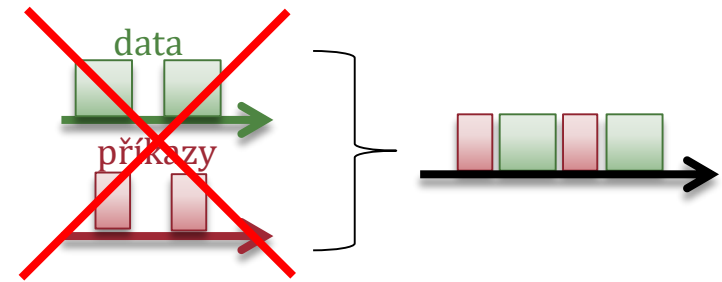
# data vs. příkazy

- **o co jde?**

- po jedné (společné) přenosové cestě přenášíme data, která mají různý význam
  - a) něco jsou „čistá data“, která mají být pouze přenesena a nesmí být měněna
  - b) něco jsou „příkazy“, které je třeba správně interpretovat (vykonat to, co požadují)

- **co je úkolem?**

- zajistit, aby vždy bylo jasné, zda jde o a) nebo b)
  - tomu se říká „zajištění transparence dat“
- a adekvátně nakládat s příkazy i čistými daty




- **kde k tomu dochází?**

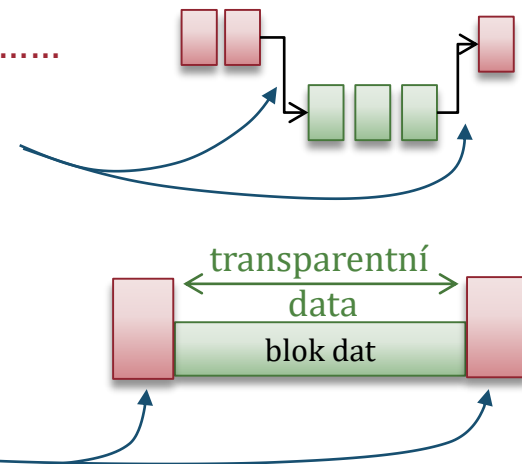
- prakticky všude
  - je to výhodnější než používat samostatné přenosové cesty pro (čistá) data a pro příkazy

- například:

- při připojování periférií k počítačům – tiskáren, modemů, myší, .....
- řeší se nejčastěji pomocí „přepínání interpretací“ (escaping)

- ale zejména:

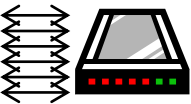
- při přenosu dat po blocích (paketech, rámcích, buňkách, ...)
  - kdy je třeba správně rozpoznat jednotlivé bloky
- řeší se pomocí tzv.  framing-u („rámování“)
  - blok dat se „zarámuje“ (vloží do vhodného ohrazení)



# příklad: ovládání modemů

## • původně:

- modemy se ovládaly pomocí samostatných signálů
  - co příkaz, to samostatný signál
  - po samostatném „drátu“
    - neúnosné, komplikované



5441324D444D314E54466B4D5455774E

```

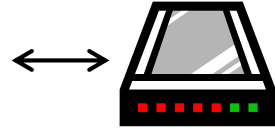
+++
OK
at&v      ← vypiš údaje o nastavení
ACTIVE PROFILE:
E0 L1 M1 Q0 T V1 X4 &C1 &D2 &G0 &P1
S00:000 S01:000 S02:043 S03:013 S04:010
S05:008 S06:004 S07:060 S08:002
S10:014 S12:050 S29:010
ati3      ← vypiš údaje o sobě
Conexant - Ambit SoftK56 V.90 (V.92) MDC
Modem
atdp123456 ← vytoč číslo pulzní volbou
BUSY
ath       ← zavěs
OK
atdt123456 ← vytoč číslo tónovou volbou
CONNECTED 9600
7A497A4E544D774D7A6B7A4D .....
  
```

přepnutí

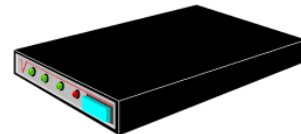
přepnutí

## • později:

- modem má jediný vstup pro data i příkazy
- modem má dva režimy:
  - datový
    - v tomto režimu chápe data na svém vstupu jako „čistá data“ a přenáší je
  - příkazový
    - v tomto režimu interpretuje data na vstupu jako příkazy a vykonává je

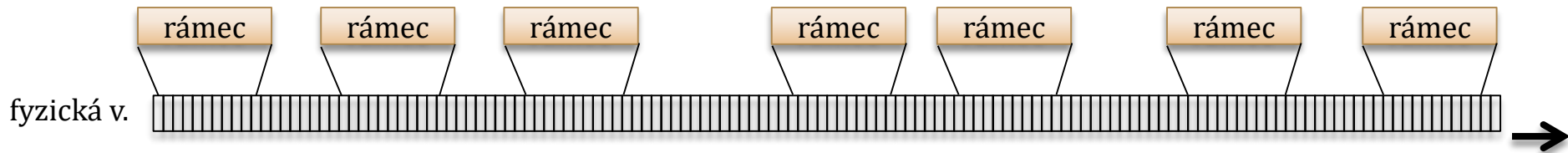


- musí existovat mechanismy pro přechod (přepnutí) mezi stavy/interpretacemi
  - z datového do příkazového režimu převádí posloupnost „+++“, následovaná sekundovou prodlevou
  - do datového režimu modem přechází automaticky po navázání spojení
- musí existovat řídicí jazyk, pro zadávání příkazů v příkazovém režimu
  - jazyk AT (od: Attention)



- zavedla firma Hayes u svého Smartmodemu (1981)

# framing (frame encapsulation)

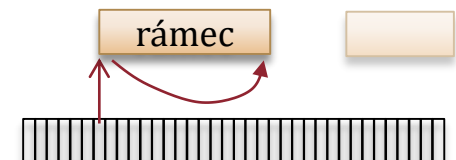
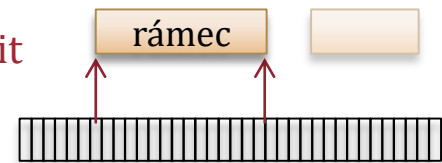
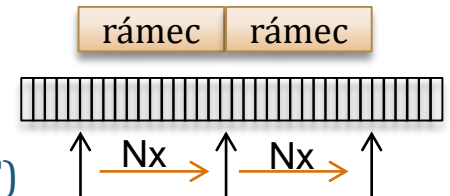


- **aneb:**

- správné rozpoznání začátku a konce přenášeného bloku dat (rámce)
- jde o úkol, který se týká hlavně linkové vrstvy
  - protože ta dostává od fyzické vrstvy dále nestrukturovaný proud bitů/bytů/znaků

- **principiální možnosti řešení:**

- **odpočítávat jednotlivé byty**
  - pokud mají rámce pevnou velikost a následují hned po sobě
    - například v telekomunikacích u digitální hierarchie (okruhů E a T)
- **vyznačit začátek a konec**
  - pomocí speciálních znaků či značek (sekvencí bitů) explicitně vyznačit začátek a konec rámce
    - **výhoda:** rámce mohou být různě velké
  - pomocí „poruch“ v kódování přenášených dat (vícehodnotová logika)
- **vyznačit začátek a odpočítat délku rámce**
  - explicitně vyznačit začátek rámce, v jeho hlavičce vyznačit velikost (počet bytů) rámce



- .....

# character, byte a bit stuffing

- potřebujeme-li vyznačit v proudu přenášených dat určitý bod

- začátek linkového rámce, nebo jeho konec

- můžeme k tomu využít různé techniky:

- character stuffing

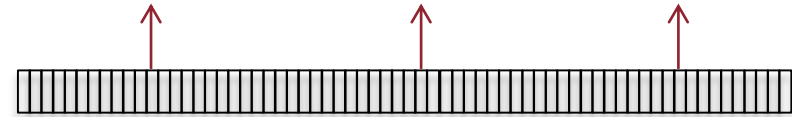
- proud přenášených dat tvoří jednotlivé znaky, začátek či konec rámce se vyznačí speciálním znakem, transparence dat se řeší vkládáním (stuffing) celých znaků

- byte stuffing

- proud přenášených dat tvoří jednotlivé byty (nikoli znaky), začátek či konec se vyznačí speciálním bytem, transparence dat se řeší vkládáním jednotlivých (celých) bytů

- bit stuffing

- proud přenášených dat tvoří jednotlivé bity, začátek či konec se vyznačí pomocí speciální sekvence bitů (tzv. křídlové značky, anglicky: flag), transparence se řeší vkládáním jednotlivých bitů



u všech variant je nutné zajistit, aby se speciální znak/byte či značka nevyskytl v těle rámce (mezi užitečnými daty)

- transparence dat

- podle toho, jakou techniku používají, se linkové protokoly dělí na:

- znakově orientované

- používají techniku character stuffing

starší protokoly (např. IBM BySync), dnes se již nepoužívají

- bitově orientované

- používají techniku byte stuffing nebo bit stuffing

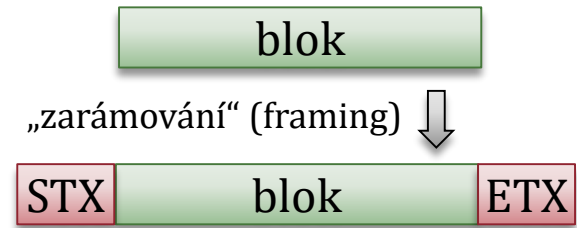
novější, dnes používané protokoly (např. HDLC, Ethernet, PPP)

# znakově orientované linkové protokoly

- k „rámování“ využívají speciální řídicí znaky

- nejčastěji: řídicí znaky v sadě ASCII:

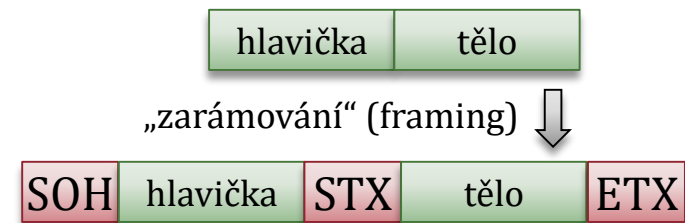
- STX (Start of TeXt): pro začátek textu/těla bloku
- ETX (End of TeXt): pro konec textu / těla bloku



- je možné i rozlišit strukturu bloku

- na hlavičku a tělo, pomocí dalších řídicích znaků

- SOH (Start Of Header): pro začátek hlavičky



- problém:

- co kdyby se některý z řídicích znaků (STX, SOH, ETX) vyskytoval v samotném bloku dat (v hlavičce nebo v těle rámce)?

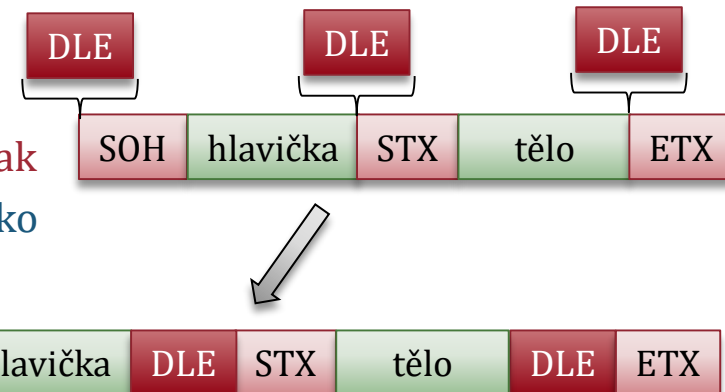
- jak potom poznat, kdy se výskyt znaku má považovat za řídicí, a kdy nikoli?

jak zajistit  
transparenci  
dat?

- řešení transparency dat:

- řídicí znaky se prefixují speciálním řídicím znakem z ASCII sady

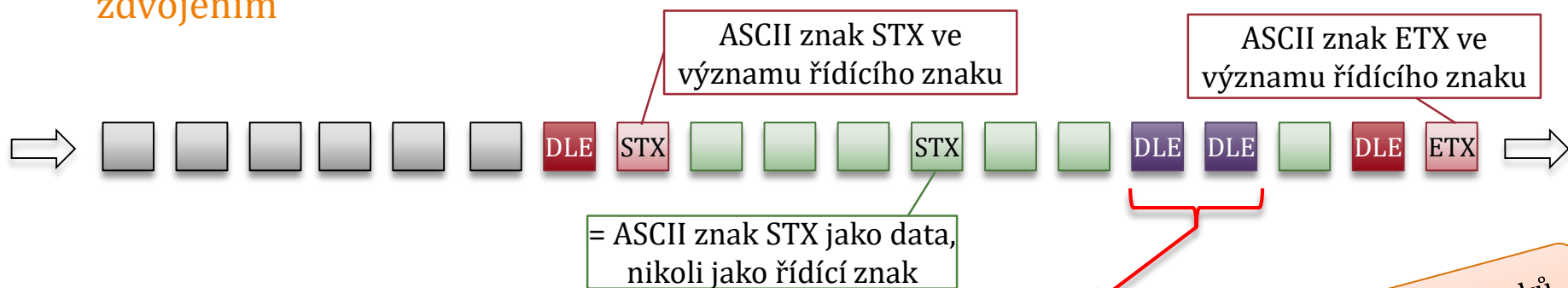
- DLE (Data Link Escape): „únikový“ (ESCAPE) znak
- význam: následující znak je třeba interpretovat jako řídicí



# znakově orientované linkové protokoly

## • řešení transparency dat (pokračování ....)

- případný výskyt speciálního řídicího znaku (DLE) v užitečných datech se řeší jeho zdvojením

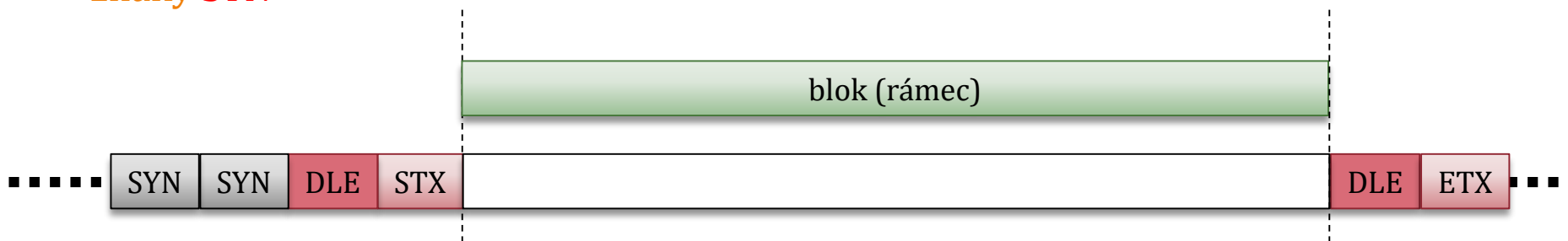


- příjemce interpreтуje dvojici znaků DLE jako 1 ASCII znak DLE
  - jako „užitečná data“, nikoli ve významu řídicího znaku

jde o techniku vkládání znaků (character stuffing)

## • řešení synchronizace

- jde o pomoc fyzické vrstvě, aby si dokázala zajistit (udržet) synchronizaci
  - aby měla „správně seřízené hodinky“ a správně vyhodnocovala bitové intervaly
- řešení u znakově orientovaných protokolů: na začátek se připojí (dva) speciální řídicí znaky SYN

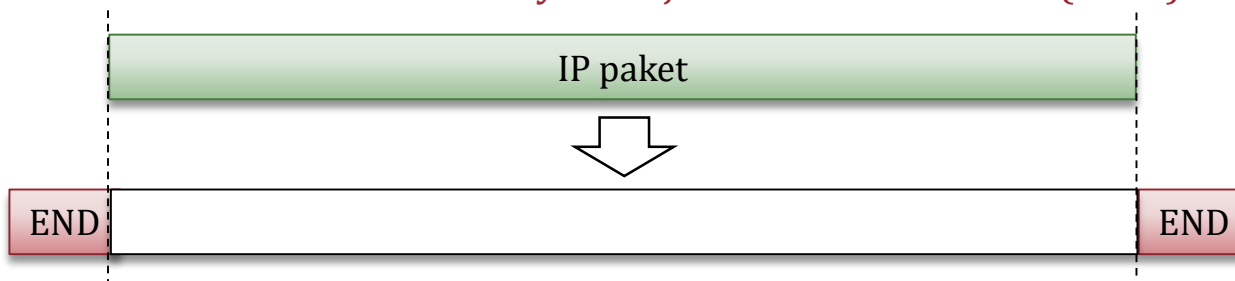
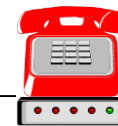




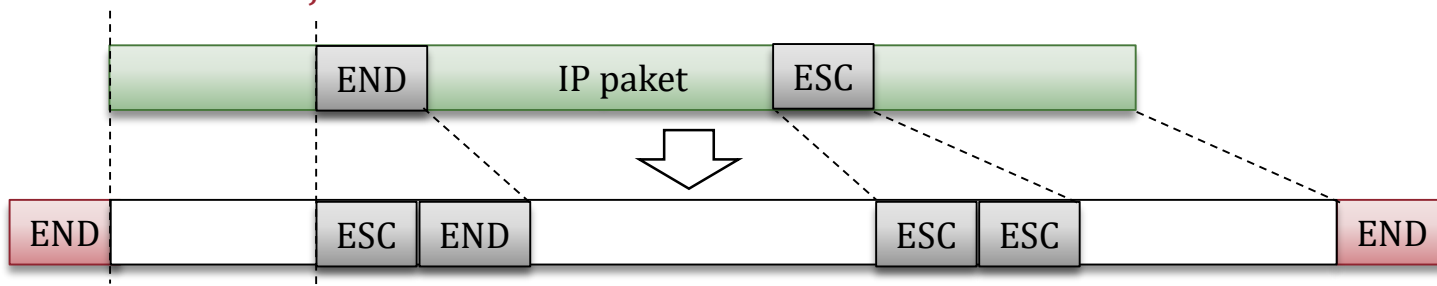
# příklad: protokol SLIP

## • SLIP: Serial Line IP

- velmi jednoduchý linkový protokol, určený pro přenos IP datagramů po dvoubodových (a plně duplexních) spojkách
  - například (po modemu) po telefonních linkách
  - nepotřebuje řešit „nic navíc“ – jen framing (vyznačení začátku a konce linkového rámce)
- je znakově orientovaný = přenášená data chápe jako posloupnost znaků
  - začátek i konec linkového rámce vyznačuje ASCII znakem **END** (0xC0)

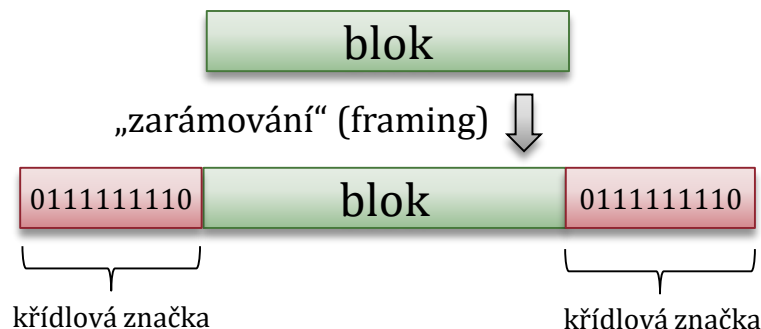


- případný výskyt řídicích znaků v těle IP datagramu řeší pomocí techniky character stuffing
  - znak **END** nahradí dvojicí znaků **ESC** (0xDB) a **END**
  - znak **ESC** nahradí dvojicí znaků **ESC** a **ESC**



# bitově orientované linkové protokoly

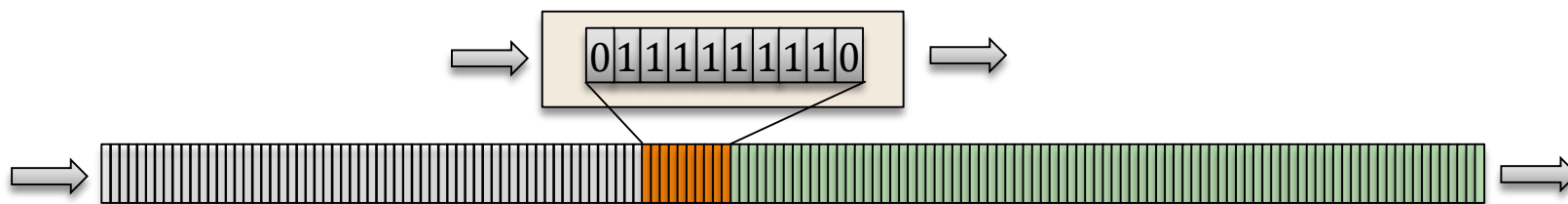
- k „rámování“ (framing-u) používají speciální posloupnost bitů
  - tzv. křídlovou značku (🇬🇧 flag)
    - nějaký předem daný „vzorek“ bitů, například určitý počet po sobě jdoucích 1 mezi krajními 0
      - jako: 0111111110 (8x 1)
- varianty:
  - křídlová značka je na začátku i na konci
    - pak je nutné zajistit, aby se nevyskytla v bloku
      - musí být vyřešena transparence dat
  - křídlová značka je pouze na začátku
    - pak je nutné nějak určit délku bloku
    - možnosti:
      - „explicitní“: délka bloku je uvedena v hlavičce
        - například u rámců Ethernet IEEE 802.3
      - „jinak“
        - například u rámce Ethernet II je konec rámce určen tím, že „skončí nosná“ (🇬🇧 carrier)
        - u kódování Manchester na 10 Mbit/s končí „časování“ (přechody mezi úrovněmi při každém bitovém intervalu, které slouží k synchronizaci)
        - a před začátkem nového rámce nosná znovu „naskočí“ (na preambuli, která slouží k synchronizaci)



# technika „bit stuffing“

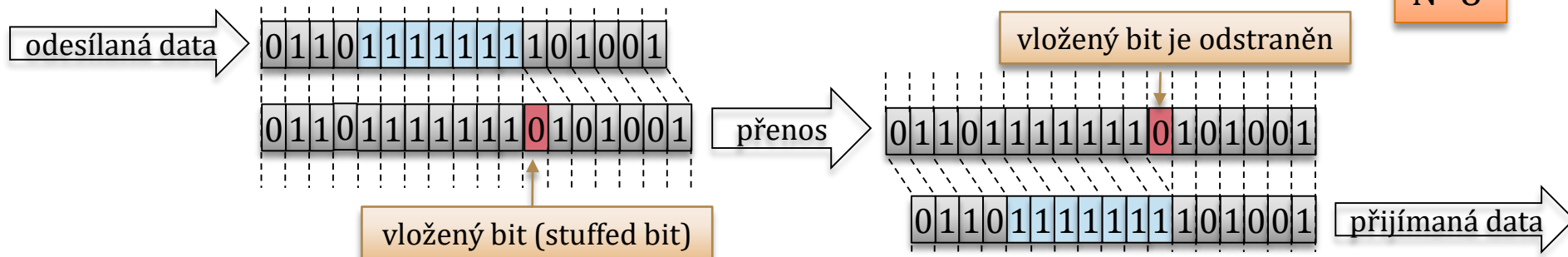
- slouží potřebám bitově orientovaných linkových protokolů
  - zajišťuje pro ně transparentci dat
    - aby se v přenášeném datovém bloku mohla vyskytovat jakákoli posloupnost bitů
      - včetně té, která jinak má význam křídlové značky
- připomenutí:
  - začátek bloku (linkového rámce) signalizuje speciální posloupnost bitů
    - tzv. křídlová značka (🇬🇧 flag), například: 011111110 (N po sobě jdoucích jedniček)

je to nutné jen tam, kde je  
koncová křídlová značka !!



- způsob fungování:
  - pokud se v (čistých) datech vyskytne posloupnost N-1 jedniček, je za ni automaticky přidána jedna nula
    - příjemce automaticky maže první nulu za posloupností N-1 jedniček

zde:  
N=8



# příklad: HDLC

## • HDLC (High-Level Data Link Control)

- příklad bitově orientovaného linkového protokolu pro spoje P-P i P-MP
- odvozený od staršího (proprietárního) protokolu SDLC firmy IBM
- standardizován od ISO (ISO 3309, ISO 4335)
- je základem/inspirací řady dalších linkových protokolů
  - PPP, LLC (Ethernet), LAPD (ISDN), LAPB (X.25), Frame Relay

## • HDLC rámec

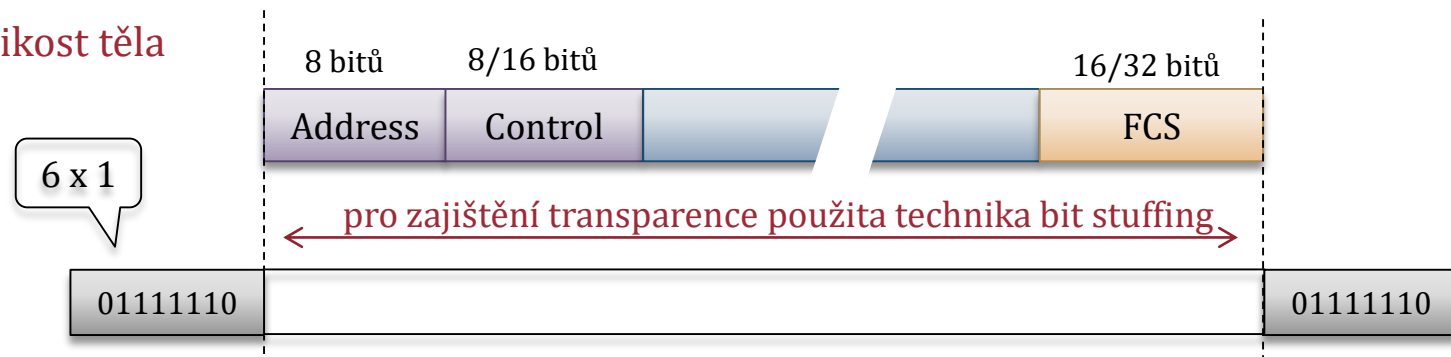
- linkový rámec protokolu HDLC
  - pevná velikost hlavičky
    - Address + Control
      - Address je adresa příjemce
  - pevná velikost patičky
    - FCS (Frame Check Sequence)
      - kontrolní součet
  - proměnná velikost těla

## • HDLC framing („rámování“)

- používá 2 křídlové značky


- počáteční a koncovou
- 6 jedniček mezi 0

stejný způsob „rámování“ (HDLC framing) mohou používat i jiné protokoly

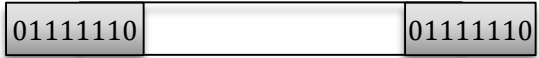


# bytově orientované protokoly

- **určitý kompromis mezi bitově a znakově orientovanými protokoly**
  - používají křídlovou značku
    - stejně jako bitově orientované protokoly
    - tato značka ale musí mít velikost (jednoho, nebo několika) bytů
      - někdy této značce předchází více (stejných) bytů pro synchronizaci příjemce
  - pokud používají křídlovou značku i pro vyznačení konce rámce, používají byte stuffing
    - pro zajištění transparence dat vkládají do užitečných dat celé „escape“ byty
      - podobně jako znakově orientované linkové protokoly
        - nejde ale o znaky, nýbrž o byty (faktický rozdíl v tom ale není, jde pouze o rozdílnou interpretaci)
      - je to méně efektivní než u bitově orientovaných protokolů
    - případný výskyt „escape“ bytů v užitečných datech musí být ošetřen
      - zdvojením či prefixací jiným „escape“ znakem
- **příklady:**
  - **Ethernet**



    - standardní způsob „rámování“ používá křídlovou značku jen na začátku
      - proto nemusí řešit výskyt této značky v těle rámce (v užitečných datech)
  - **protokol PPP (Point-to-Point Protocol z rodiny TCP/IP)**



    - používá křídlovou značku na začátku i na konci rámce
      - musí řešit transparenční dat – pomocí techniky byte stuffing (vkládáním celých bytů)

# příklad: Ethernet

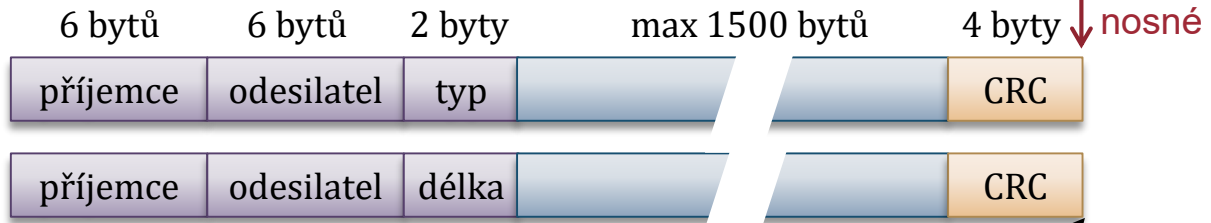
## Ethernet má vlastní formát linkových rámců

- dokonce 4 různé formáty: Ethernet II, IEEE 802.3+802.2, SNAP a „raw“

– formát Ethernet II:

– formát IEEE 802.3:

nepotřebuje bit stuffing



## Ethernet má i vlastní framing

– pro synchronizaci používá tzv. preambuli

- posloupnost 7 bytů s hodnotou 0x55 (10101010)

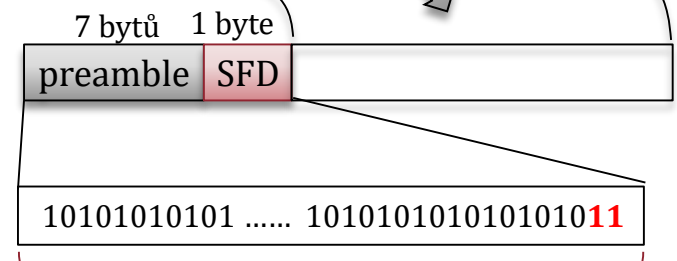
– pro synchronizaci

– a tzv. Start Frame Delimiter (SFD) křídlová značka

- 1 byte o hodnotě 0xD5 (přenášeno jako 10101011)

– pořadí bitů: konvence Little Endian

– pořadí bytů: konvence Big Endian

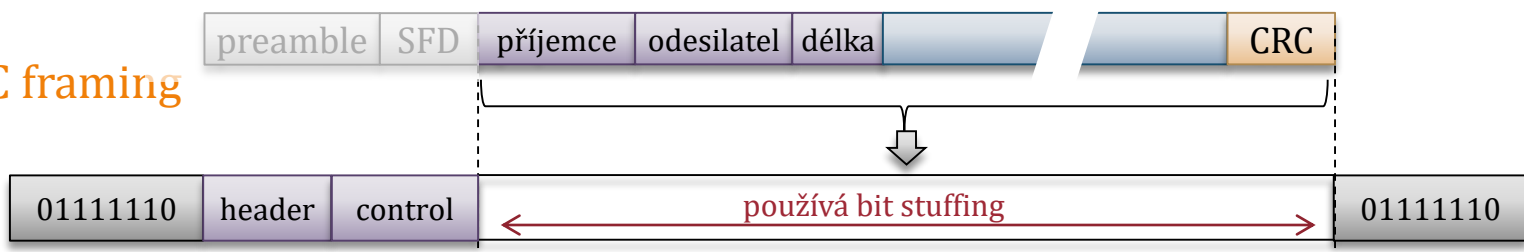


posloupnost pravidelně se střídajících 0 a 1 (sinusovka), se „změnou fáze“ na konci

## ale může používat (dříve používal) i jiný framing

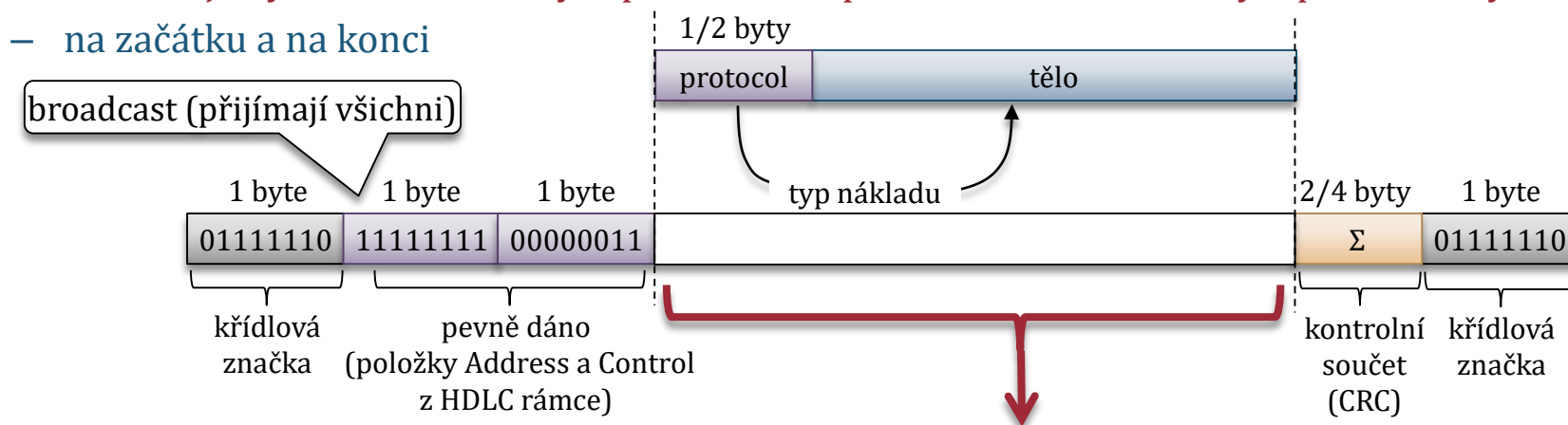
– např. HDLC framing

potřebuje bit stuffing

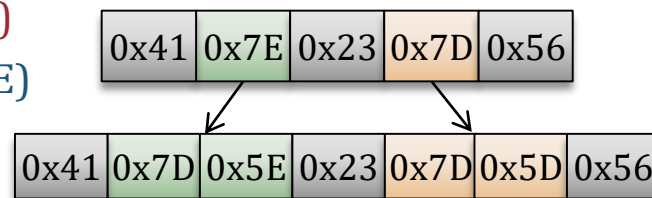


# příklad: PPP (Point to Point Protocol)

- jde o linkový protokol z rodiny TCP/IP, určený pro dvoubodové spoje
  - lze do něj vkládat pakety různých síťových protokolů
    - v praxi: především IP pakety
- je odvozen od protokolu HDLC
  - standardně používá framing („rámování“), převzatý z protokolu HDLC
    - kvůli tomu je bytově orientovaným protokolem: používá křídlové značky v podobě 1 bytu



- pro zajištění transparency dat používá techniku byte stuffing
  - křídlová značka je 01111110 (6x1, 2x0), tj. 0x7E
  - případný výskyt křídlové značky v datech (tj. nikoli v řídicím významu) je prefixován bytem s hodnotou 0x7D (posloupností 01111101, resp. znak ESC)
    - původní křídlová značka je XOR-ována s 0x20 (na 0x5E)
  - případný výskyt znaku 0x7D (ESC) v datech je zdvojen
    - původní ESC znak je opět XOR-ován s 0x20 (na 0x5D)



# zajištění spolehlivosti - přehled

- **připomenutí:**

- zajištění spolehlivosti může být požadováno po různých vrstvách
  - příklad: v TCP/IP se řeší až na transportní vrstvě (protokol TCP)
  - v ISO/OSI je požadováno již po síťové vrstvě (a může být požadováno i po linkové vrstvě)

- **otázka:**

- co všechno je nutné pro zajištění spolehlivosti datových přenosů?

1. schopnost detekovat ztrátu celých bloků

- řeší se skrze počítání bloků (rámců, paketů, zpráv, ....)
- nebo skrze potvrzování pozice přenesených dat v bytovém proudu (protokol TCP)

2. schopnost detekovat změnu (chybu) v bloku přenesených dat

- řeší se pomocí mechanismů detekce chyb (parita, kontrolní součty, CRC)

3. schopnost nápravy

- ad 1: je nutno řešit **opakovaným přenosem** ztracených dat

- předpokladem je možnost **potvrzování** (acknowledgement), umožňující vyžádat si opakování přenosu konkrétního bloku dat

- ad 2: lze také řešit **opakovaným přenosem** celého bloku (poškozených) dat

- nebo: pokusit se o **(samo)opravu** chybných (poškozených) dat

nejsou  
nikdy  
100% !!

je k tomu nutná velká  
redundance  
(např. použití Hammingových  
kódů apod.)

- **v praxi:**

- možnost samoopravy je nákladná a využívá se jen minimálně
  - jen tam, kde neexistuje zpětná vazba a možnost vyžádat si opakování přenosu



# jak řešit detekci chyb?

- chyby mohou být různé:
  - **pozměněná data**
    - některé (jednotlivé) bity jsou změněny
  - **shluky chyb**
    - celé větší skupiny bitů/bytů jsou změněny
  - **výpadky dat**
    - celé bloky dat jsou ztraceny
- co má smysl detekovat?
  - pokud se náprava řeší opakovaným přenosem celého bloku, je zbytečné zjišťovat:
    - kolik chyb je v daném bloku, jaké druhu jsou a kde přesně v rámci bloku k nim došlo
  - protože stejně se bude daný blok přenášet znovu
- proto:
  - stačí detekovat chyby s přesností na celé bloky (pakety, rámce, zprávy, ....)
    - ve smyslu: **daný blok je bez chyb** / **daný blok je chybný**

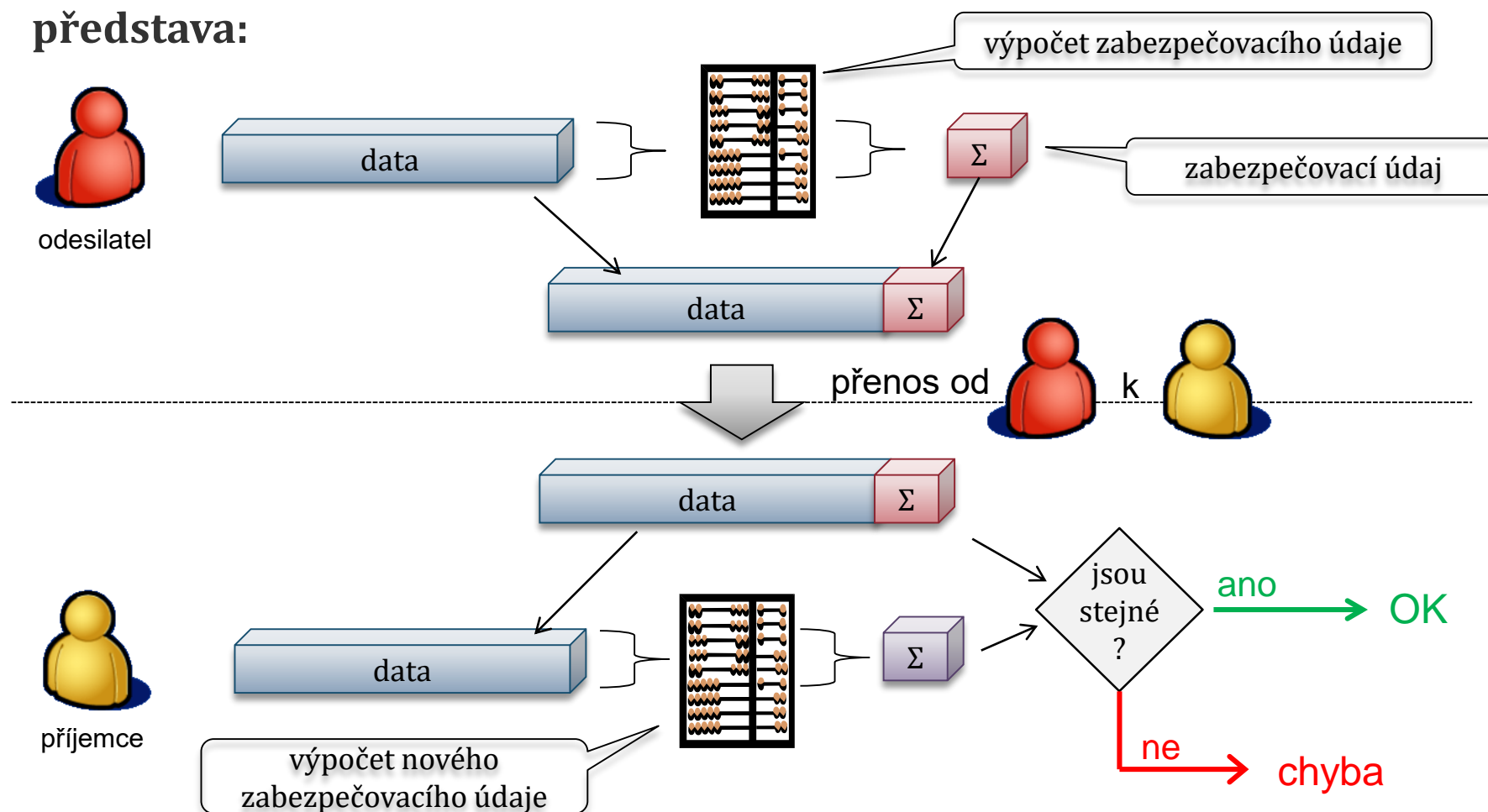
skrze potvrzování se  
odesilatelí potvrdí úspěšný  
přenos bloku

skrze potvrzování se vyžádá  
jeho opakovaný přenos

v očekávání, že to již  
dopadne dobře

# zabezpečení přenášených bloků

- mechanismy zabezpečení (parita, kontr. součty, CRC) mají stejnou podstatu
  - k přenášeným blokům dat přidávají „zabezpečovací údaj“
    - paritní bity, kontrolní součet, výsledek po dělení CRC polynomem
  - příjemce využije tento „zabezpečovací údaj“ k detekci, zda došlo k chybě
- představa:



# parita

## • parita (paritní bit)


- je bit přidán navíc k datovým bitům
- **sudá parita:**
  - paritní bit je nastaven tak, aby celkový počet 1 byl sudý
- **lichá parita:**
  - ..... aby byl lichý
- **existuje také:**
  - **jedničková parita:**
    - paritní bit je pevně nastaven na 1
      - nemá zabezpečující efekt
  - **nulová parita**
    - ... nastaven na 0

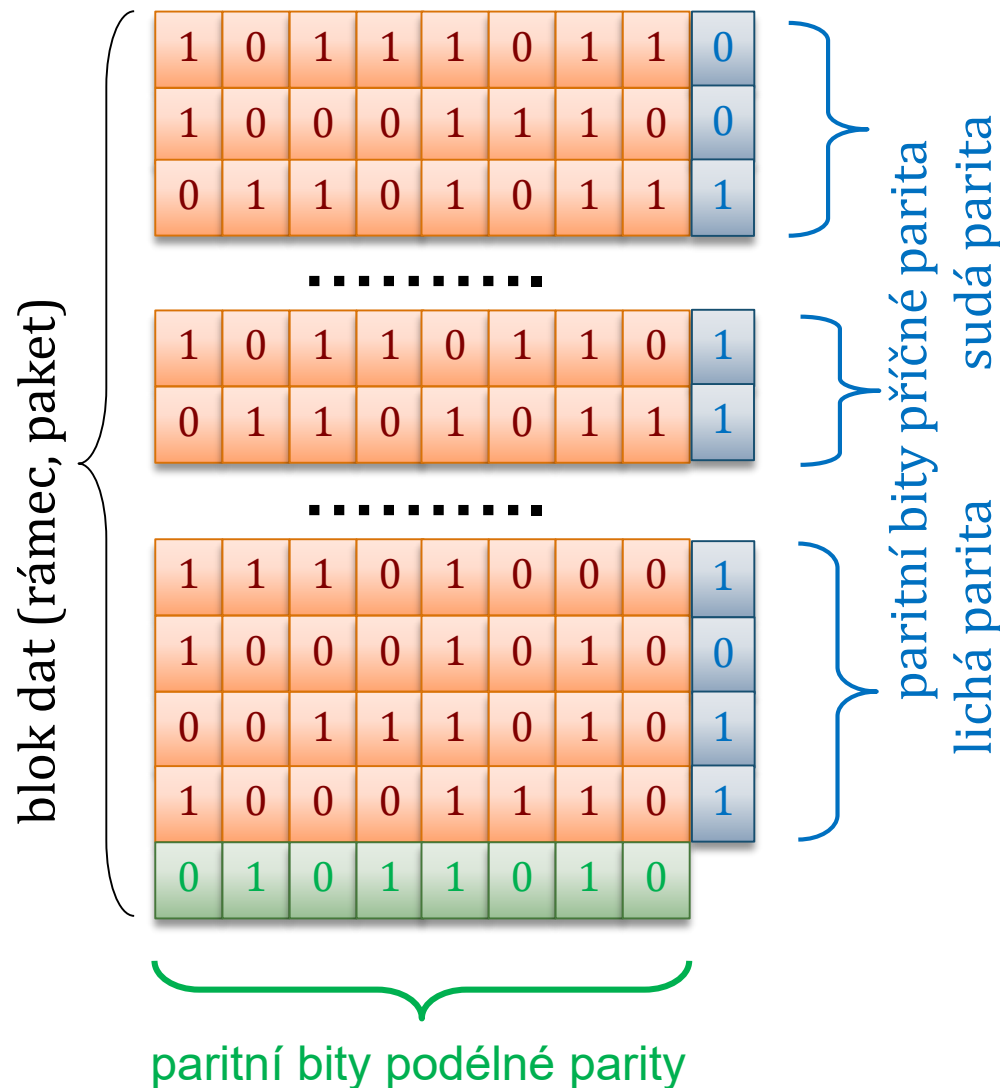


## • **podélná parita:**

- je počítána ze všech stejnohlých bitů všech bytů/slov

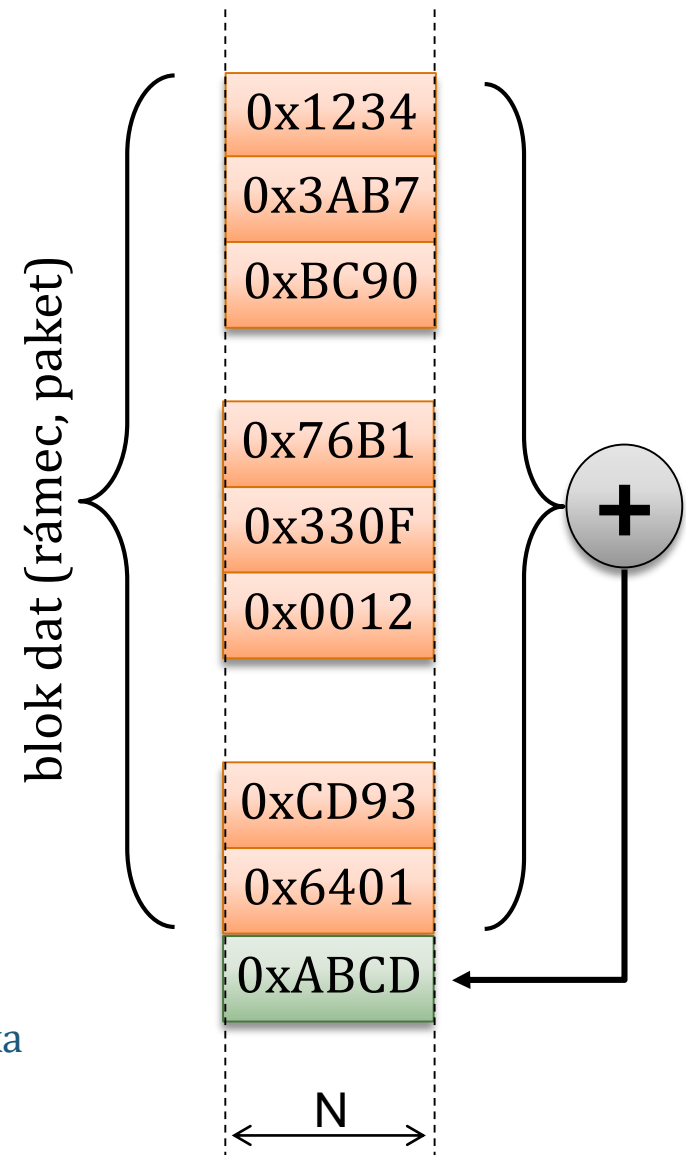
## • **příčná parita:**

- je počítána po jednotlivých bytech/slovech

  - informace o tom, který byte (slovo) je poškozen, je ale nadbytečná
    - stejně se znovu posílá celý blok



# kontrolní součet

- **nevýhoda parity**
  - má malou „účinnost“ – odhalí jen „drobné“ chyby
    - například chyby v jednotlivých bitech
      - zatímco chyby ve dvou bitech se mohou vzájemně „vyrušit“
  - účinnější je kombinace příčné a podélné parity
- **kontrolní součet**
  - je „účinnější“ než parita
    - dokáže detekovat více chyb
      - ale také není zdaleka „dokonalý“ – řadu chyb nedetekuje
  - princip:
    - blok dat, určený k přenosu, se interpretuje jako posloupnost bytů/slov
    - jednotlivé byty/slova této posloupnosti se sečtou
      - alternativa: dělá se jejich XOR
    - výsledný součet se použije jako zabezpečovací údaj
      - přesněji: použije se zbytek modulo N, kde N je šířka bytu/slova



# CRC – Cyclic Redundancy Check

- posloupnost bitů, tvořící blok dat, je interpretována jako polynom

- přesněji: jednotlivé bity tvoří koeficienty polynomu nad tělesem charakteristiky 2

$$\dots + 1 \cdot x^{14} + 0 \cdot x^{13} + 0 \cdot x^{12} + 1 \cdot x^{11} + \dots$$

... 1 0 1 0 0 1 0 0 ...

- takovýto polynom je vydělen jiným polynomem

- tzv. **charakteristickým polynomem**
  - např. (CRC-16):  $x^{16} + x^{15} + x^2 + 1$

- schopnosti detekce jsou vynikající

- tímto způsobem je možné detekovat:

- všechny shluky chyb s lichým počtem bitů
- všechny shluky chyb do velikosti N bitů
  - kde N je stupeň charakteristického polynomu
- všechny shluky chyb velikosti  $> N+1$  s pravděpodobností 99.99999998%
  - pro CRC-32

musí být volen velmi pečlivě (složitě)

- výsledkem je podíl a zbytek

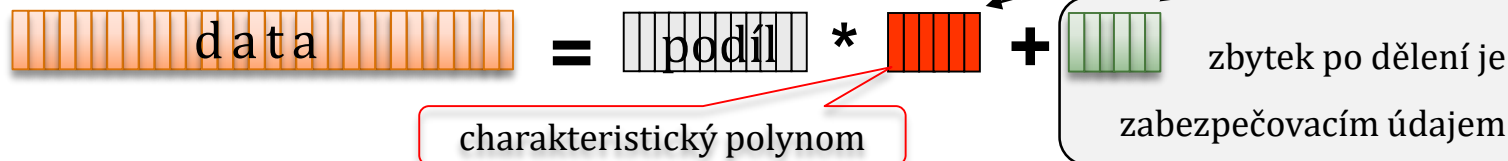
- v roli zabezpečení se použije zbytek po dělení charakteristickým polynomem
- chápaný již jako posloupnost bitů

- přitom

- složitost (implementace) výpočtu je minimální

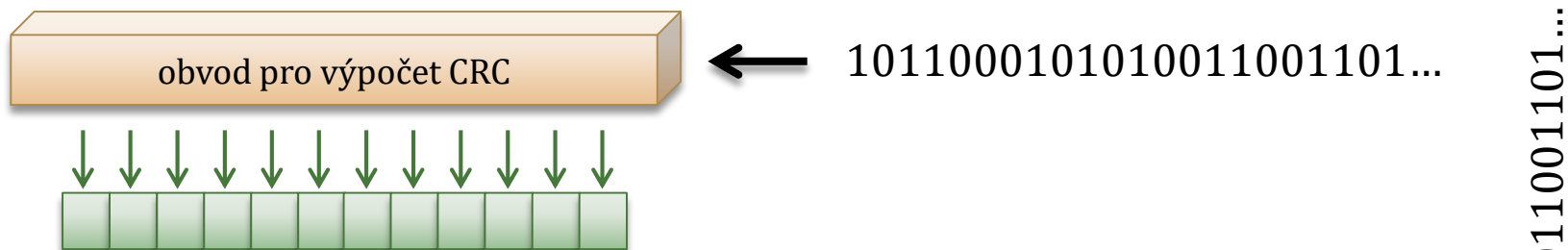
- lze snadno „zadrátovat“

v rozsahu 8/16/32/64/... bitů



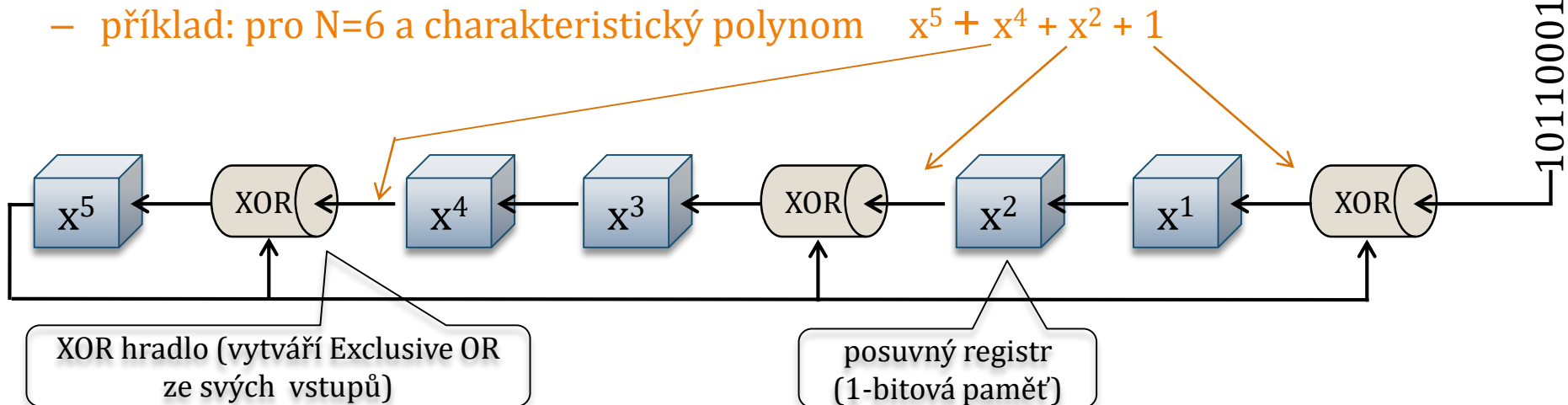
# příklad: obvod pro výpočet CRC

- spolehlivost CRC kódů se opírá o silné teoretické výsledky z algebry
- ale:
  - samotný výpočet CRC-kódu (zbytku po dělení) je velmi jednoduchý
    - může být snadno implementován v HW, pomocí XOR-hradel a posuvných registrů
  - jde o jednoduchý „sériový“ obvod, který postupně přijímá jednotlivé bity




- na konci (postupného vkládání jednotlivých bitů) zůstane v obvodu (v jeho posuvných registrech) zbytek po dělení, který slouží jako zabezpečovací údaj

- příklad: pro  $N=6$  a charakteristický polynom  $x^5 + x^4 + x^2 + 1$



# potvrzovací strategie

- **snaha napravit chybu/ztrátu dat opakováním přenosu (retransmission)**
  - očekávání: podruhé k problémům již nedojde
    - což nemusí být splněno – k chybě či ztrátě může dojít znovu
- **podmínkou je:**
  - možnost, aby příjemce dal najevo odesílateli, že má data odeslat znovu (celý blok)
  - řeší se pomocí **potvrzování (acknowledgement)**
-  **společně je vše označováno jako ARQ (Automatic Repeat reQuest)**
  - jde o potvrzovací strategii (či „potvrzovací schéma“)
    - fakticky jde o celý protokol pro zajištění spolehlivosti přenosu
- **existuje více různých potvrzovacích strategií (ARQ):**
  1. **jednotlivé potvrzování (Stop&Wait ARQ)**
    - využívá jednotlivé potvrzování a před odesláním dalšího bloku čeká na (kladné) potvrzení předchozího bloku
  2. **kontinuální potvrzování s návratem (Go-Back-N ARQ)**
    - využívá kontinuálního potvrzování a při opakování přenosu (retransmission) se vrací k místu, kde k chybě/ztrátě došlo
  3. **kontinuální potvrzování se selektivním opakováním (Selective Repeat ARQ)**
    - využívá kontinuálního potvrzování a při opakování přenosu se nevrací, ale znovu přenesení jen poškozený/ztracený blok

# jednotlivé potvrzování



## Stop&Wait ARQ

### • princip:

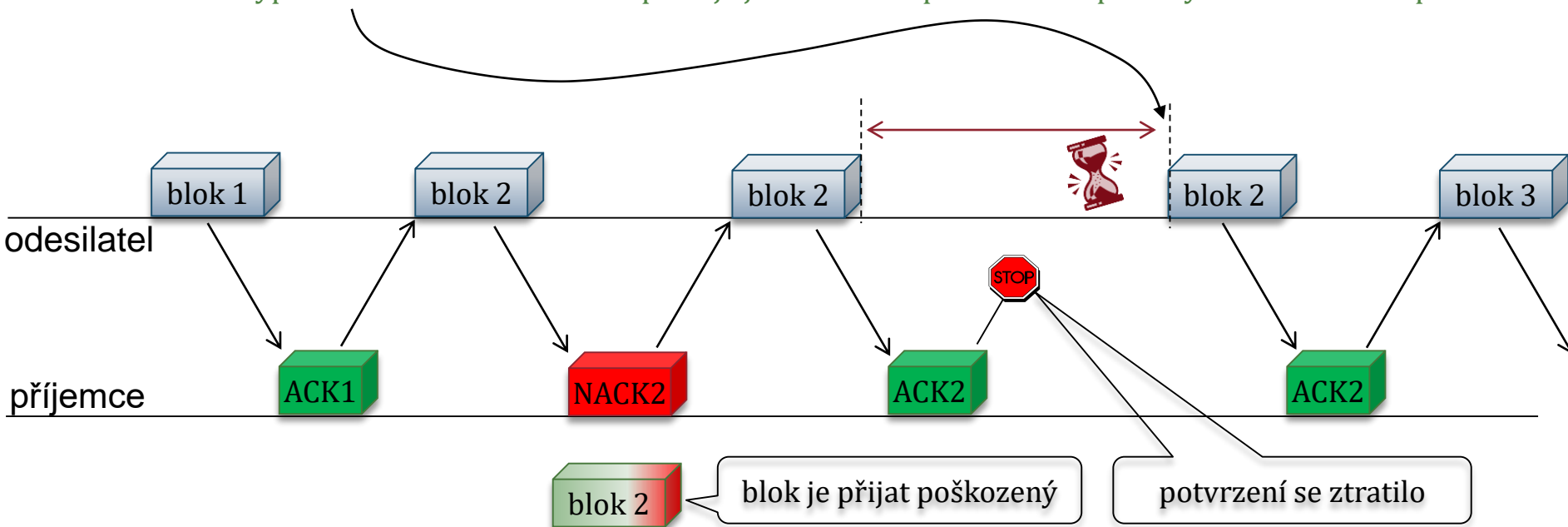
#### 1. každý blok je potvrzován jednotlivě

- buď pomocí **kladného potvrzení** (ACK – blok byl přijat bez detekované chyby), nebo
- pomocí **záporného potvrzení** (Negative ACK, NACK – blok byl přijat s chybou)
  - a je nutné opakovat přenos



#### 2. po odeslání každého bloku se odesílatel zastaví (STOP) a čeká (WAIT) na potvrzení

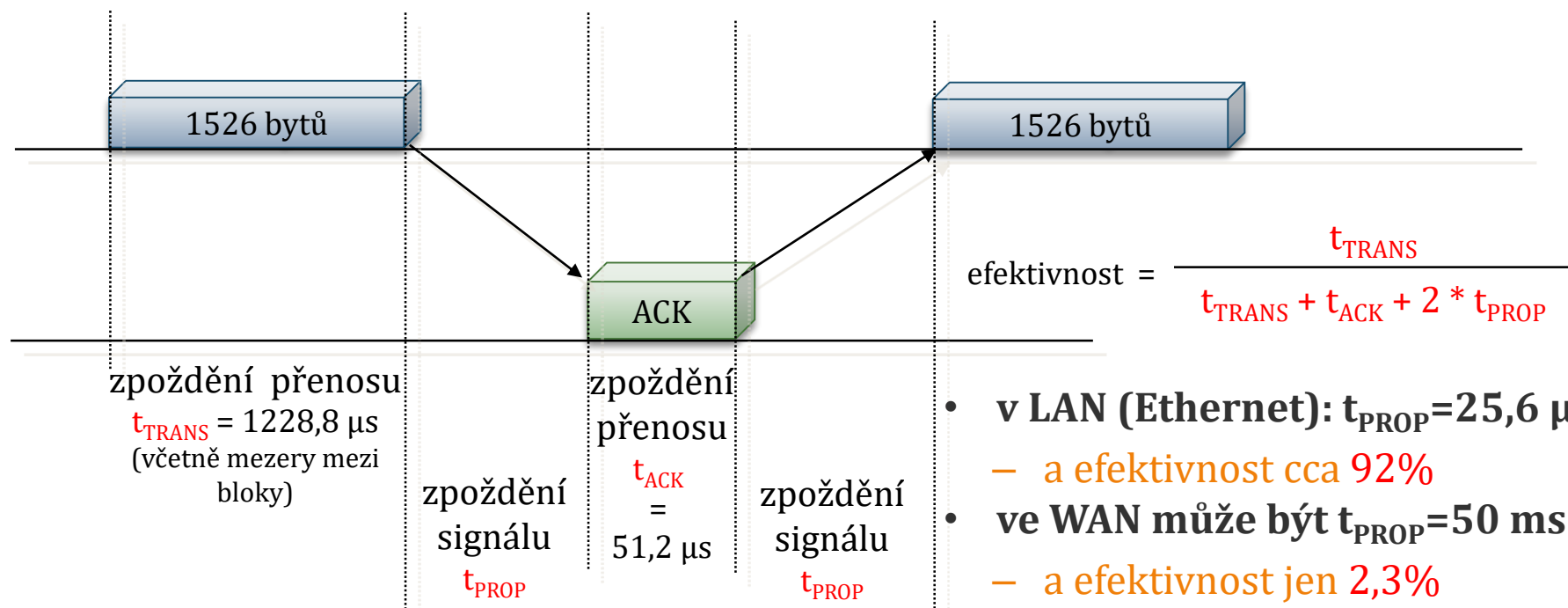
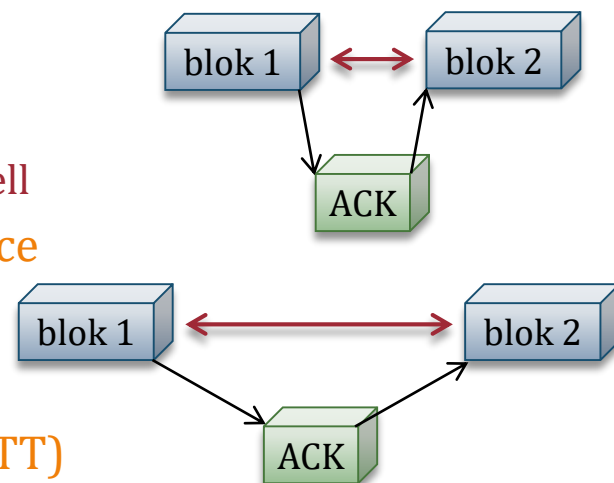
- další blok odešle až po přijetí kladného potvrzení (ACK)
- v případě přijetí záporného potvrzení (NACK) znovu odešle původní blok
  - to samé v případě, že do doby T (časového limitu / timeout-u) nedostane žádné potvrzení
    - vypršení časového limitu interpretuje jako ztrátu či poškození naposledy uskutečněného přenosu





# vlastnosti jednotlivého potvrzování

- **jednoduchá a „přímočará“ strategie**
  - snadno se implementuje
    - byla použita např. v protokolech IPX/SPX společnosti Novell
  - jejím důsledkem je polo-duplexní charakter komunikace
    - nikdy se nepřenáší oběma směry současně
- **nevýhody:**
  - je neefektivní v sítích s větší latencí/dobou obrátky (RTT)
  - příklad: 10 Mbit/s Ethernet



# kontinuální potvrzování

- **problém jednotlivého potvrzování:**

- **hodí se jen do lokálních sítí**

- nikoli do sítí rozlehlých

- **alternativa pro rozlehlé sítě**

- **nečekat na potvrzení, ale ihned posílat další bloky**

- tedy: ještě dříve, než přijde potvrzení předchozích bloků
- vlastně: bloky se posílají souvisle, kontinuálně (continuously)

- proto: **kontinuální potvrzování (Continuous ARQ)**

- potvrzení přichází „až později“ (zpětně) .....

- **praktické aspekty:**

- **co dělat, když nějaké potvrzení nepřijde vůbec (nebo je negativní)?**

- mezitím již mohlo být odesláno více dalších bloků .....

- **možná řešení:**

- **kontinuální potvrzování s návratem (Go-Back-N ARQ)**

- odesílání se „vrací zpět“ do místa, kde došlo k poškození/ztrátě, dále se pokračuje (znovu) od tohoto místa

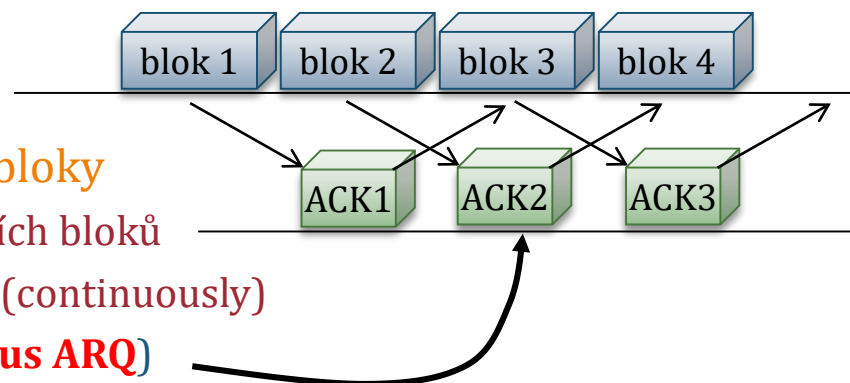
- tj. některé již odeslané bloky se odesílají znovu

- **selektivní opakování (Selective Repeat ARQ)**

- odešle se pouze („selektivně“) ten blok, který byl poškozen či ztracen, pak se pokračuje, jako kdyby k žádné chybě nedošlo



## Continuos ARQ



používá protokol TCP z TCP/IP

TCP používá standardně

TCP „umí“ volitelně

# kontinuální potvrzování s návratem

## • princip:

- poškozený/ztracený blok se přenesení znovu
- a po něm se postupně přenáší následující bloky
  - které již mohly být odeslány (a třeba i úspěšně doručeny)

### Go-Back-N ARQ

přenos se „vrací zpět“  
k poškozenému/ztracenému bloku

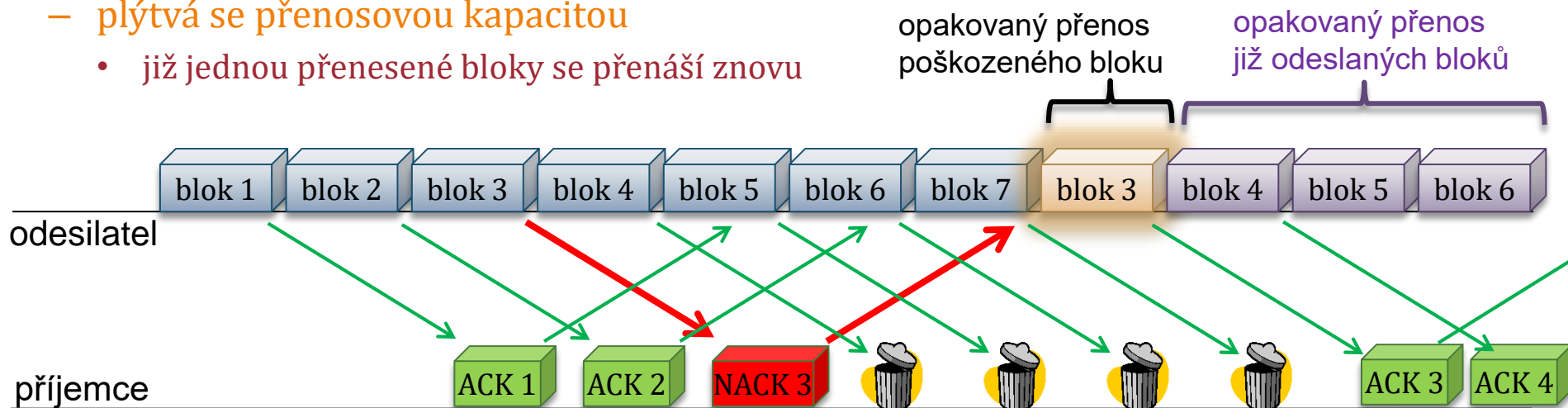
přenáší se znovu ....

## • výhody:

- je to jednodušší na implementaci
- pro příjemce je to jednodušší
  - když přijme poškozený blok (nebo mu některý blok „vypadne“ kvůli ztrátě), nemusí ukládat další bloky do bufferů a pouze čeká na opakované zaslání poškozeného/ztraceného bloku
    - protože další bloky „dostane“ znovu

## • nevýhoda:

- plýtvá se přenosovou kapacitou
  - již jednou přenesené bloky se přenáší znovu



# selektivní opakování

## Selective Repeat ARQ

opakuje se (selektivně) pouze přenos poškozeného/ztraceného bloku

- princip:**

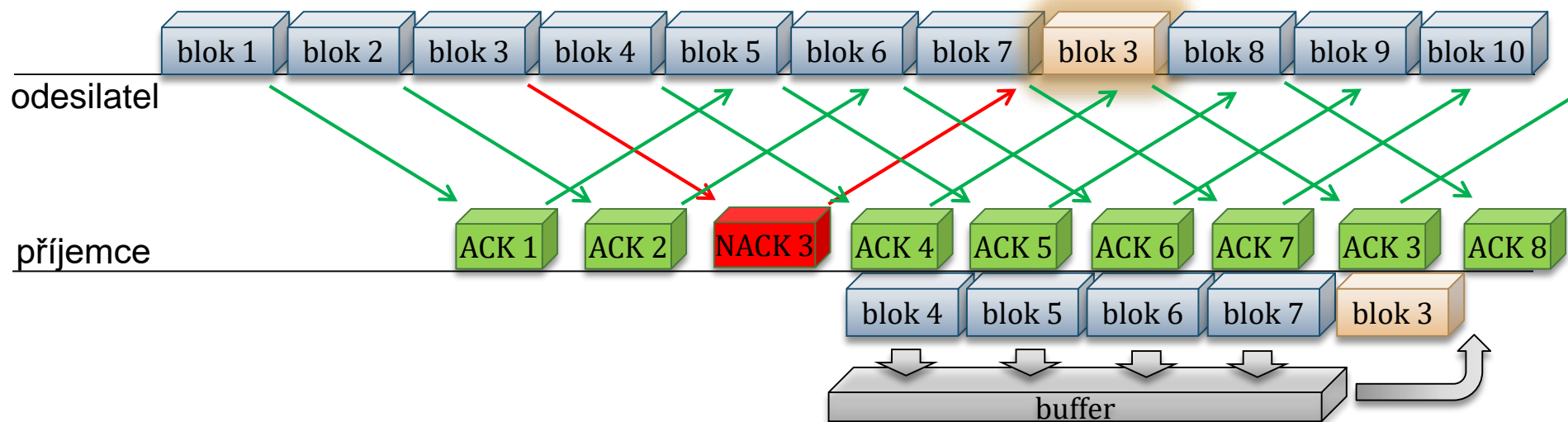
- poškozený/ztracený blok se přenese znovu
- dále se pokračuje, jako kdyby k žádné chybě/ztrátě nedošlo
  - jsou přenášeny ty bloky, které ještě nebyly přeneseny

- výhody:**

- neplýtvá se přenosovou kapacitou (jednotlivé bloky dat se nepřenáší zbytečně)

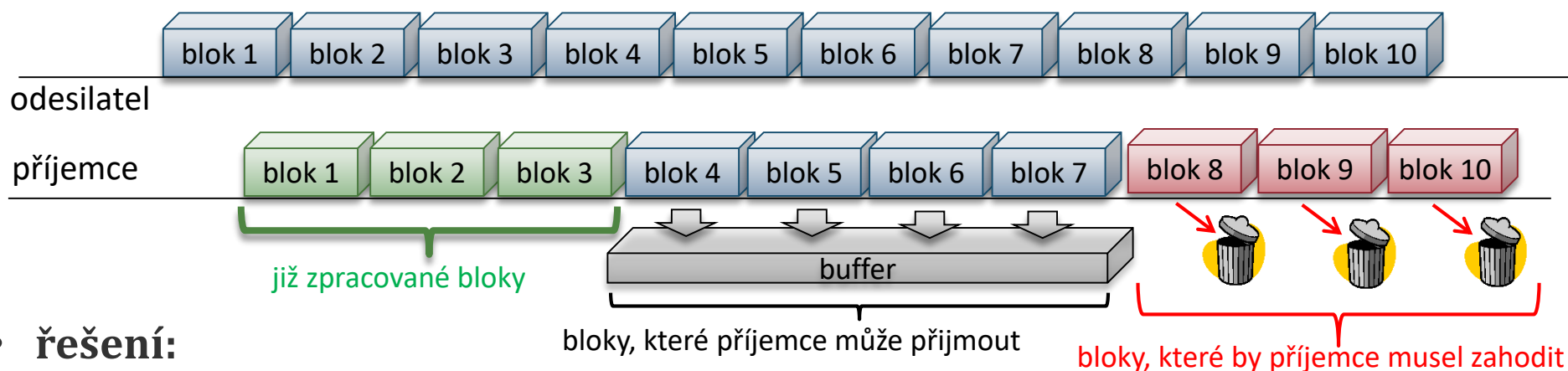
- nevýhoda:**

- pro příjemce je to náročnější
  - bloky, přenášené po poškozeném/ztraceném bloku, musí ukládat „do zásoby“ (do bufferů), ale ještě je nemůže zpracovávat
  - se zpracováním musí čekat na opakovaný přenos poškozeného/ztraceného bloku

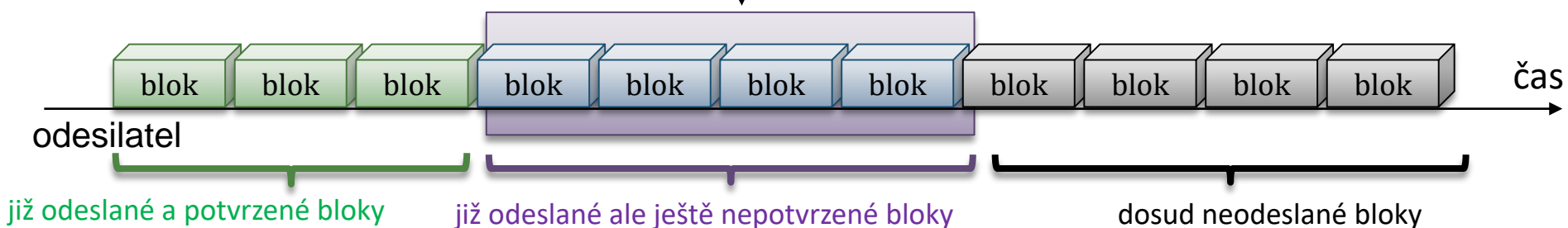


# metoda (posuvného) okénka

- kontinuální potvrzování umožňuje:
  - odesílat bloky dat maximálním možným tempem
    - dopředu, ještě než přijde jejich potvrzení
- to nemusí být dobře!!
  - příjemce nemusí mít dostatečnou kapacitu na zpracování přijatých bloků
    - a může být nucen je zahazovat !!



- řešení:
  - odeslat „dopředu“ (bez potvrzení) jen tolik bloků, kolik příjemce dokáže přijmout
    - podle velikosti „okénka“



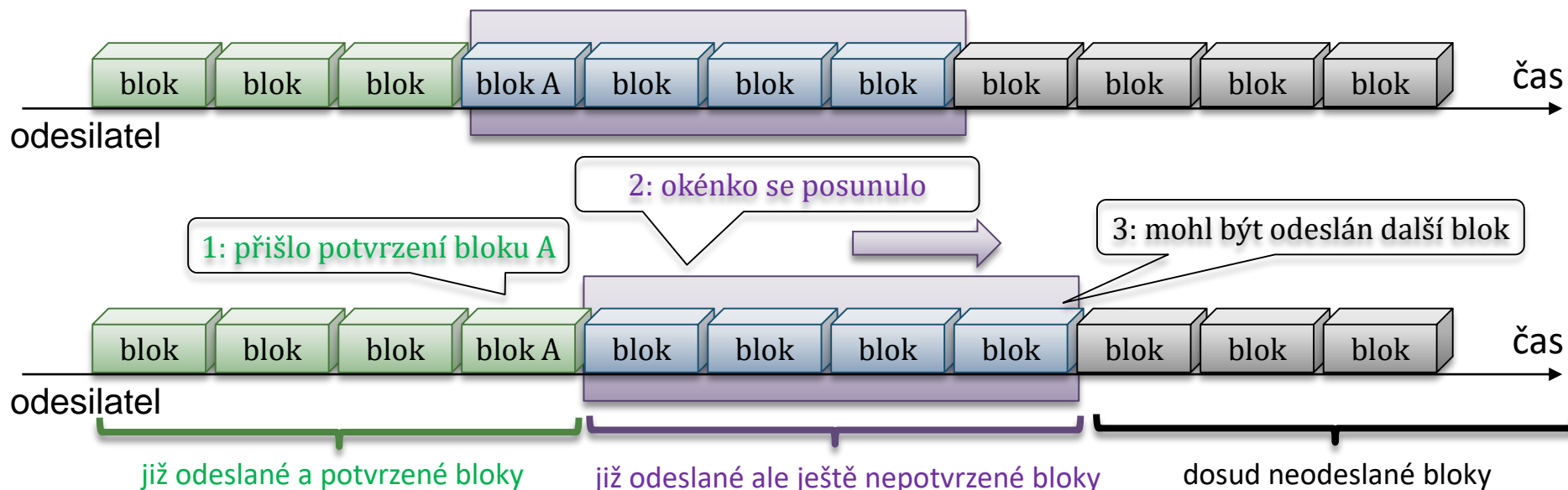
# metoda (posuvného) okénka



sliding window

## • proč „posuvné“?

- protože s příchodem (kladného) potvrzení se okénko posouvá



## • otázka:

- jak volit (optimální) velikost okénka?

## • odpověď:

- velikost okénka může stanovovat odesílatel
  - např. podle toho, jak a kdy mu přichází jednotlivá potvrzení
- velikost okénka může stanovovat i příjemce
  - podle svých možností

výsledná velikost je minimem  
z obou hodnot

řeší to problém řízení toku

# řízení toku

**flow control**

- jde o řešení obecného problému:

- jak nezahltit příjemce?

- podstata problému:

- odesílatel může být mnohem “výkonnější” než příjemce

- příjemce nemusí „stíhat“ tempo, které odesílatel dokáže vyvinout

- předpokládá se, že propojení (sít') mezi odesílatelem a příjemcem má dostatečnou kapacitu a problém neovlivňuje

- sít' není úzkým hrdlem v komunikaci mezi odesílatelem a příjemcem

- princip řešení:

- odesílatel se při odesílání řídí kapacitními možnostmi příjemce

- možnosti praktického řešení:

- řízení toku může být implementováno na různých vrstvách

- fyzická/linková:

- příjemce dává najevo odesílateli, zda má či nemá odesílat

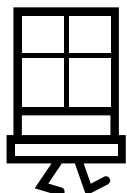
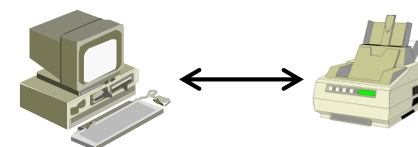
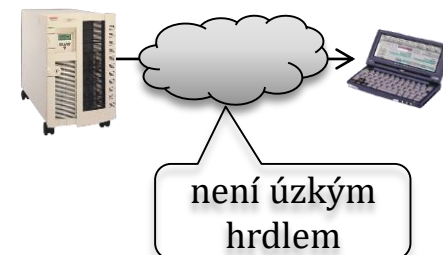
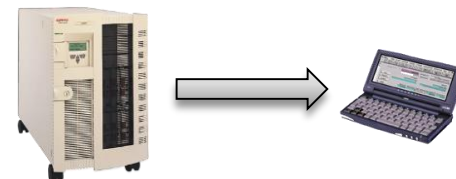
- pomocí signálů: RTS/CTS, nebo pomocí řídicích znaků: XON/XOFF

- vyšší vrstvy a metoda okénka:

- příjemce (spolu)určuje maximální velikost posuvného okénka

- v rámci potvrzení „inzeruje“ novou velikost okénka – tím říká, kolik dalších dat je schopen přijmout

- takto to funguje v protokolu TCP na transportní vrstvě



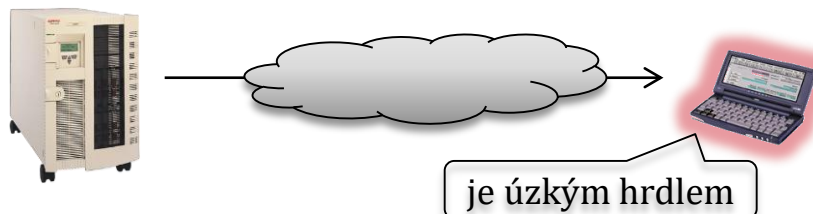
# předcházení zahlcení

 congestion control

- jde o řešení jiného problému, než u řízení toku

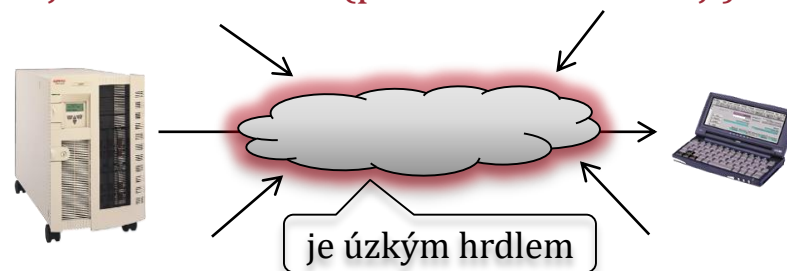
- řízení toku:

- problémem je disproporce mezi kapacitními možnostmi odesílatele a příjemce
- jejich propojení (sít') do problému nevstupuje – má dostatečnou kapacitu



- předcházení zahlcení:

- problémem je kapacita propojení (sítě) mezi odesílatelem a příjemcem
  - může být nedostatečná a může dojít k zahlcení této sítě
  - tato síť může být zatěžována také dalšími přenosy, které probíhají souběžně !!!!
- kapacitní možnosti odesílatele a příjemce jsou dostatečné (problém neovlivňují)



- řešení (jak předcházet zahlcení):

- dopředné techniky

- snaží se ovlivňovat to, co se posílá do sítě
  - neposílat do sítě takové datové toky, které by způsobily zahlcení
    - konkrétně: techniky „upravování provozu“ (traffic conditioning)

- zpětnovazební techniky

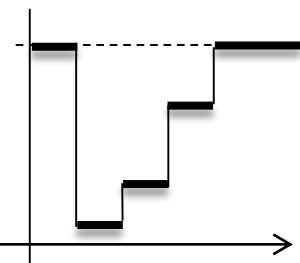
- snaží se zpětně reagovat na příznaky zahlcení (či explicitní zprávy/hlášení o zahlcení)
  - omezují další vysílání



# techniky předcházení zahlcení

## • zpětnovazební techniky

- potřebují se nějak dozvědět, že došlo k zahlcení přenosové sítě
  - potřebují nějakou formu zpětné vazby
- možnosti:
  - odesílatel dostane explicitní informaci
    - v TCP/IP: ICMP zpráva Source Quench
      - jednostranný „výkřik“ od směrovače, že se blíží/již nastalo jeho zahlcení
      - moc se nepoužívá
  - dedukce: odesílatel si nějak domyslí, že došlo k zahlcení sítě
    - protokol TCP: když nedostane včas potvrzení, interpretuje to jako zahlcení
      - které způsobil právě on !!!!
      - reaguje přechodem na režim „slow start“
        - přejde na jednotlivé potvrzování a pouze postupně zvětšuje své okénko
        - tj. nejprve odešle jeden blok, pak čeká na potvrzení, pak odešle dva bloky a čeká na jejich potvrzení, pak čtyři bloky atd.



## • dopředné techniky

- nepotřebují žádnou zpětnou vazbu
- snaží se upravovat ten provoz, který teprve vstupuje do přenosové sítě
  - v očekávání, že k zahlcení nedojde
  - obecně: tzv. **traffic conditioning**
- pracují s určitým předpokladem o tom, co síť ještě „unes“
  - aniž by došlo k jejímu zahlcení
    - jakoby: zná „laťku“
- možnosti:
  - **traffic shaping**
    - provoz, který je „nad laťku“, se snaží pozdržet a přenést později



## • traffic policing

- provoz, který je „nad laťku“, ihned zahazuje

