

# PA159, přednáška 2

21. a 28. 9. 2007

# Co nás dnes čeká...

- Úvaha o tom, jak zapisovat protokoly
- Úvod do abstraktní protokolové notace
- Zajištěné přenosové protokoly
- TCP

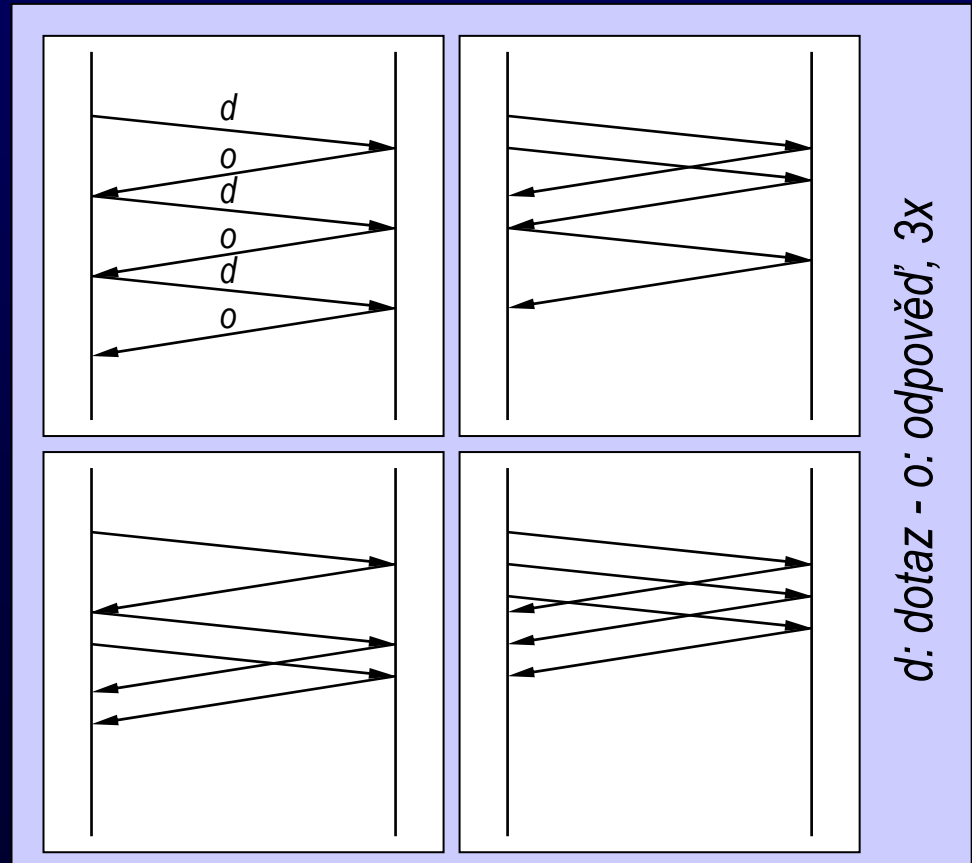
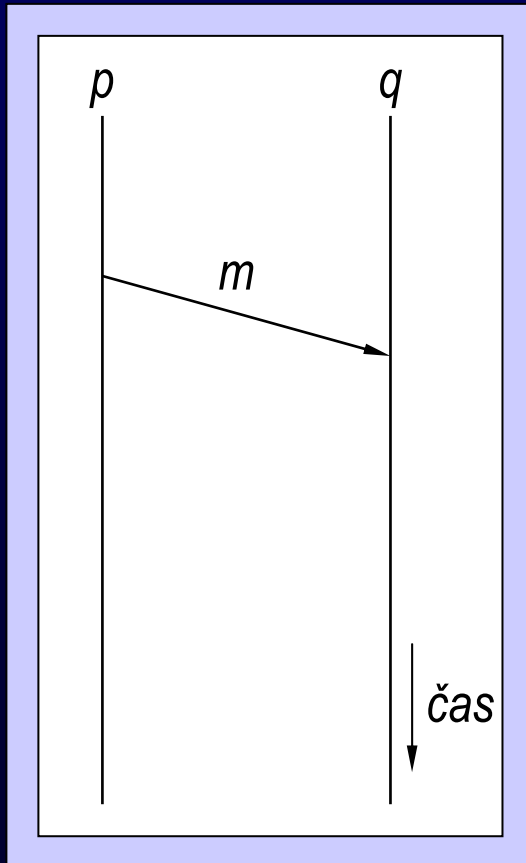
# Abstraktní protokolová notace

# Specifikace protokolů

- Jak vůbec formálně protokol specifikovat?
- Popis v přirozeném jazyce
  - problém s exaktností, jednoznačností
- Popis v programovacím jazyce
  - typicky v C
  - neohrabané, matematicky nepraktické a neelegantní
  - hůře čitelné (na druhé straně notaci zná dost lidí)
  - problém s verifikací
- Diagramy

# Specifikace protokolů (2)

- Časové osy



$d$ : dotaz -  $o$ : odpověď, 3x

# Specifikace protokolů (3)

- Časové osy
  - přehledné pro jednoduché případy, obtížně použitelné pro složitější
- Problém s dokazováním/verifikací
- => Potřeba formální notace
  - Abstraktní protokolová (AP) notace
    - M. Gouda, *Protocol verification made simple: a tutorial*. Computer Networks and ISDN Systems 25 (1993), 969 - 980
    - M. Gouda, *Elements of Network Protocol Design*, 1998



# Elementy AP

- Základní struktura specifikace

```
process <process name>

const   <const name>, ... , <const name>
inp     <inp name>:<inp type>, ... , <inp name>:<inp type>
var     <var name>:<var type>, ... , <var name>:<var type>

begin
    <action> [] ... [] <action>
end
```

- Typy konstant
  - positive integer
  - sdílení konstant se stejným jménem přes procesy



# Elementy AP (2)

- Typy vstupů
  - boolean, range, integer, set, enumerated, array
  - nesdílí se

```
inp    <inp name> : boolean
inp    <inp name> : <lower bound> .. <upper bound>
inp    <inp name> : integer
inp    <inp name> : set {<member>|<condition>}
inp    <inp name> : <name of an input of type set>
inp    <inp name> : array [<index range>] of <type>
```

- Proměnné
  - boolean, range, integer, enumerated, array
  - nesdílí se

# Elementy AP (3)

- Akce

```
<guard of p> -> <statement of p>
```

- Stráže

- operátory

- porovnání: =, /=, <, >, <=, >=
    - logické operace:  $\wedge$ ,  $\vee$ ,  $\sim$
    - aritmetické operace: +, -, \*,  $+_n$ ,  $-_n$ , min, max

- příjem dat

```
recv <message of p> from <process name q>
```

- timeout

```
timeout <protocol predicate>
```

# Elementy AP (4)

- Výrazy

- dolce far niente

```
skip
```

- přiřazení

```
x.0, ..., x.k := E.0, ..., E.k
```

- *napřed se vyhodnotí, pak se přiřadí!*

- odeslání zprávy/dat

```
send <message p> to <process name q>
```

# Elementy AP (5)

- sekvence výrazů

```
<statement>; <statement>
```

- podmíněný výběr ( $\bigvee_k \text{local\_guard.k} = \text{true}$ )

```
if <local_guard.0> -> <statement_0> [] ...  
    [] <local_guard.k> -> <statement.k> fi
```

- opakování

```
do <local_guard> -> <statement> od
```

- Kanál  $p \rightarrow q$ :  $ch.p.q$ 
  - $\#ch.p.q$  - počet zpráv v kanálu  $ch.p.q$
  - $m\#ch.p.q$  - počet zpráv  $m$  v kanálu  $ch.p.q$

# Elementy AP (6)

- Vykonávání specifikace protokolu
  - *nedeterministický výběr*
    - výběr z více povolených možností ve výrazu if
    - výběr z více povolených akcí
  - *atomicita akce*
  - *férové vykonávání protokolu*
    - je-li akce neustále povolena, bude určitě vykonána
    - definice přes nekonečné běhy

# A nyní trochu praxe...

- Kódování Manchester
  - odesílající proces

```
process s

inp data : array [0..1] of integer

var x : integer

begin
    true -> if data[x] = 0 ->      send one to r;
                                     send zero to r
          [] data[x] = 1 ->      send zero to r;
                                     send one to r;
          fi;
          x := x + 1;
end
```

## – přijímající proces

```
process r

var rcvd : array [0..1] of integer
    y : integer,
    store : boolean,      {store := false}
    lastb : 0..1

begin
    rcvd zero from s ->
        if store ^ lastb = 1 ->
            rcvd[y], y, store := 0, y+1, false
        [] ~store v lastb = 0 ->
            store := true;
        fi; lastb = 0
    rcv one from s ->
        if store ^ lastb = 0 ->
            rcvd[y], y, str := 1, y+1, false
        [] ~store v lastb = 1 ->
            store := true;
        fi; lastb = 1
    rcv error from s ->    store := ~store;

end
```

# Elementy AP (7)

- zprávy s poli

```
<message name> ( <field.0>, ..., <field.n )  
send m(x.0,..., x.n) to q  
recv m (x.0,..., x.n) from p
```

- nedeterministické přiřazování

```
x := any  
x := random
```

- pole procesů

```
process <process array> [<index> : <type>, ..., <index> : <type>]  
process p[i: 0..n-1]  
send mesg to p[i+1]  
recv mesg from p[i-1]
```



# Elementy AP (8)

- parametrizované akce
  - v deklarační části přibude

```
par <parameter declaration list>
```

- použití

```
par j : 0..2, r : 0..1  
...  
requested[j,r] -> requested[j,r] := false; send grant(r) to p[j]
```

```
requested[0,0] -> requested[0,0] := false; send grant(0) to p[0]  
[] requested[1,0] -> requested[1,0] := false; send grant(0) to p[1]  
[] requested[2,0] -> requested[2,0] := false; send grant(0) to p[2]  
[] requested[0,1] -> requested[0,1] := false; send grant(1) to p[0]  
[] requested[1,1] -> requested[1,1] := false; send grant(1) to p[1]  
[] requested[2,1] -> requested[2,1] := false; send grant(1) to p[2]
```

# A opět trochu praxe...

- protokol pro alokaci zdrojů
  - uživatel

```
process user[i: 0..n-1]

const   s {# of resources}
var     wait : boolean,
         r : 0..s-1

begin
    ~wait ->           wait, r := true, any;
                     send request(r) to controller;
[]         rcv grant(r) from c -> wait := false;
                     send release(r) to controller
end
```

## - hlídač

```
process controller

const  s {# of resources}, n
var    avail : array [0..s-1] of boolean,
       requested : array [0..n-1, 0..s-1] of boolean
par    j : 0..n-1,
       r : 0..s-1

begin
  rcv request(r) from u[j] -> requested[j,r] := true
[] rcv release(r) from u[j] -> avail[r] := true
[] avail[r] ^ requested[j,r] -> avail[r], requested[j,r] :=
                                false, false;
                                send grant(r) to user[j];
end
```

- protokol není férový - hlídač může stále ignorovat jeden proces (pozor, podmínka férového výkonu protokolu sice platí, ale potíže je v tom, že díky nastavení `avail[r] := false` je příslušná akce na vypnuta!)

- řešení: a) determinismus (klientům přiděluji v definovaném pořadí)
- b) randomizace:

```

process controller

const   s {# of resources}, n
var     avail : array [0..s-1] of boolean,
         requested : array [0..n-1, 0..s-1] of boolean
         k : 0..n-1,
         x : 0..n-1 {random user}
par     j : 0..n-1,
         r : 0..s-1

begin

    rcv request(r) from u[j] ->
        if avail[r] ->   avail[r] := false;
                        send grant(r) to user[j]
        [] ~avail[r] ->  requested[j,r] := true

[]      rcv release(r) from u[j] ->
        x := random;
        k := x +n 1;
        do k /= x  $\wedge$  ~requested[k,r] -> k := k +n 1 od;
        if requested[k,r] ->   requested[k,r] := false;
                                send grant(r) to user[k]
        [] ~requested[k,r] -> avail[r] := true
        fi

end

```

# Chyby při přenosu sítí

- 3 základní typy
  - ztráta dat
  - poškození dat
    - zahozením se dá převést na ztrátu
  - přeuspořádání zpráv/paketů
- pro snazší práci se zavádí 2 pravidla
  - atomičnost chyb
  - chyby se vyskytují zřídka (pouze konečný počet chyb v potenciálně nekonečném běhu protokolu)

# Ošetření chyb

- příjem poškozených dat
  - převedeme na ztrátu

```
recv error from p -> skip
```

- ztráta paketu => timeout

```
timeout
  ~ready
  ^ (sendmoney#ch.client.machine +
     getselection#ch.machine.client = 0)
```

# Normální timeouty

- Forma normálního timeoutu

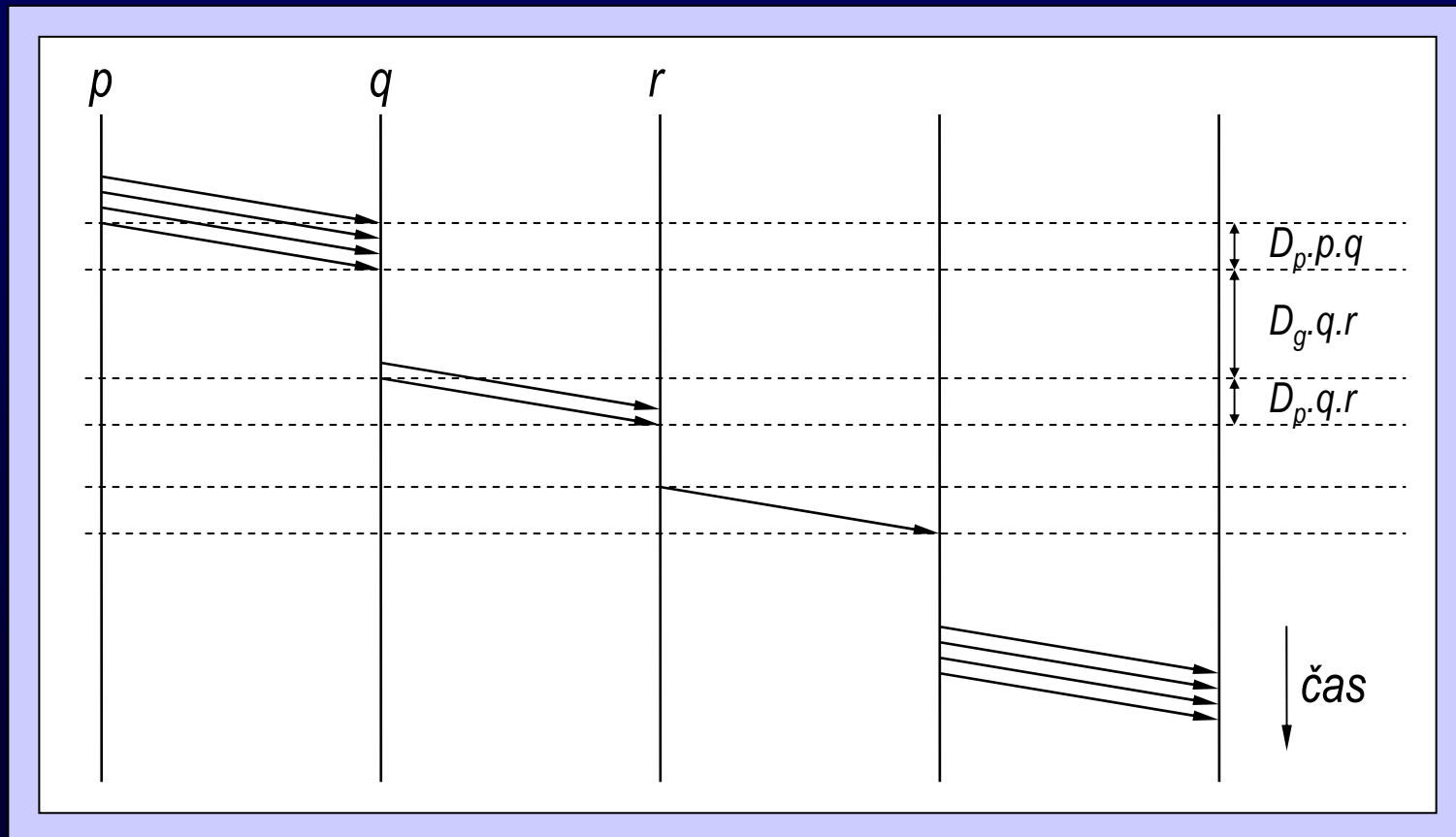
```
timeout
    (local predicate of process p)
    ∧ (mp#ch.p.q ≤ kp)
    ∧ (mq#ch.q.r ≤ kq)
    ...
    ∧ (my#ch.y.z ≤ ky)
```

- po každý pár  $(ms\#ch.s.t \leq ks) \wedge (mt\#ch.t.u \leq kt)$  platí, že stráž akce posílající zprávy  $mt$  má tvar `recv ms from s`

```
timeout
    ~ready
    ∧ (sendmoney#ch.client.machine ≤ 0)
    ∧ (getselection#ch.client.machine ≤ 0)
```

# Normální timeouty (2)

- Implementace pomocí hodin s reálným časem





# Zajištěný protokol pro transportní vrstvu

# Požadavky na protokol

- Odolnost vůči chybám
  - kontrola integrity dat
  - detekce výpadků pomocí timeoutů
  - vypořádání se s přeuspořádáním (nějak)
- Obrana proti zahlcení přijímajícího uzlu
  - informace od přijímající strany
- Obrana proti zahlcení sítě
  - není-li spolehlivá odezva ze sítě, odhad

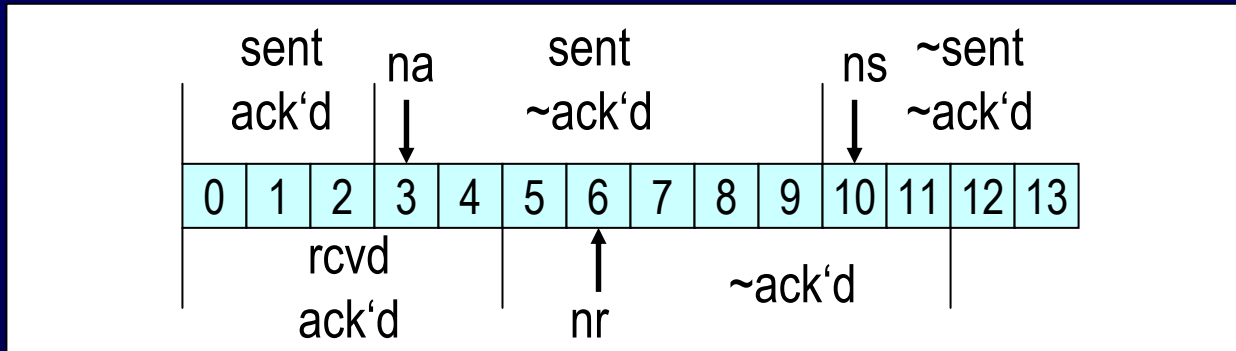
# Odolnost proti chybám

- Detekce poškození dat
  - Backward Error Recovery
    - ověření integrity
      - kontrolní součty
    - v případě selhání data skartována
    - vyžádá se nová kopie
  - Forward Error Recovery
    - data jsou posílána redundantně tak, že se omezené výpadky dají rekonstruovat
      - n-ohraničené poškození, ztráty a přeuspořádání

# Odolnost proti chybám (2)

- Detekce výpadku => potvrzování
  - vlastně také Backward Error Recovery
  - určité množství dat může být odesláno bez potvrzení
- Typy potvrzení
  - individuální potvrzování
  - kumulativní potvrzování
  - blokové potvrzování
  - negativní potvrzování (přímo informace o výpadku)

# Kumulativní potvrzování



```
process sender
```

```
const    w    {ns - na <= w}  
var     na, ns, i : integer
```

```
begin
```

```
    na + w > ns ->    send data(ns) to receiver;  
                    ns := ns + 1
```

```
[]    rcv ack(i) from receiver ->    na := max(na, i)  
[]    timeout t-guard ->            send data(na) to receiver  
[]    rcv error from receiver ->    skip
```

```
end
```

```
t-guard = na < ns  $\wedge$  (#ch.q.p = 0)
```

# Kumulativní potvrzování (2)

```
process receiver

const    w
inp      wr : integer    {receiver window}
var      nr, j : integer,
         rcvd : array [0..wr-1] of boolean,
         akn : boolean   {true if receiver hasn't acked last received message}

begin
  rcv data(j) from p ->
    if true ->          {busy receiver} skip
    [] j < nr ->        akn := true
    [] nr <= j < nr + wr ->
      rcvd[j mod wr] := true;
      do rcvd[nr mod wr] ->
        {deliver data(nr)}
        rcvd[nr mod wr], nr, akn :=
          false, nr+1, true
      od;
    fi
  [] akn -> send ack(nr) to sender; akn := false;
  [] rcv error from sender -> skip
end
```

# Kumulativní potvrzování (3)

- Je nutné použít neohraňovaná sekvenční čísla
  - příklad problému s ohraničenými čísly:
    1. odesílatel pošle data(3), data(4)
    2. příjemce pošle ack(4), ack(5)
    3. ack(5) se zadrhne na lince a je předběhnut následnými acky
    4. dojde k přetočení ohraničených sekvenčních č.
    5. přijde ack(5)
    6. neštěstí je hotovo ;-)

# Další typy potvrzování

- Individuální
  - potvrzování každé zprávy zvlášť
    - neefektivní
  - použití cirkulárního bufferu o velikosti nejméně  $2w$
  - možnost použití ohraničených sekvenčních čísel
- Blokované
  - potvrzování přijatých spojitých bloků
  - cirkulární buffer o velikosti nejméně  $2w$
  - možnost použití ohraničených sekvenčních čísel



# Řízení velikosti okna

- modifikace odesílatele u kumulativního potvrzování
  - stačí modifikace odesílatele
  - povede-li se bez ztráty zprávy odeslat více než  $c_{max}$  zpráv, je okno  $w$  inkrementováno o 1
  - dojde-li k timeoutu a zmenšení velikosti okna, je třeba počítat s tím, že mohlo dojít ke ztrátě dat v dosud nepotvrzeném okně a zmenšovat proto dál okno (condition DNR)

```

process sender

const   wmin, wmax           {window limits, wmin < wmax}
inp    cmax : integer      {cmax > 0}
var    na, ns, i : integer,
        w : wmin..wmax,      {ns - na <= w}
        c : integer,        {counter for consecutive data ack'd
                               without being resent}
        ra, rs : integer    {auxiliary variables assigned to na+1, ns
                               when packet loss occurs}

begin

    na + w > ns ->   send data(ns) to receiver;
                    ns := ns + 1
[]   rcv ack(i) from receiver ->
        na, c := max(na, i), c + max(i-na, 0)
        if c >= cmax -> w, c := min(w+1, wmax), c-cmax
        [] c < cmax -> skip
        fi
[]   timeout t-guard  $\wedge$  ~akn ->
        if (ra <= na  $\wedge$  na < rs) -> skip {condition DNR}
        [] ~(ra <= na  $\wedge$  na < rs) ->
            w, rs := max(w/2, wmin), ns
        fi; ra, c := na + 1, 0;
        send data(na) to receiver
[]   rcv error from receiver ->           skip
end

```

# Kvalita protokolu

- Agresivita
  - schopnost využít dostupnou volnou kapacitu
- Responsivnost
  - schopnost přizpůsobení se menší kapacitě
- Férovost
  - férové (případně jinak definované) dělení šířky pásma mezi klienty při požadavcích převyšujících možnosti sítě

TCP

# Z historie

- Poprvé formálně specifikován r. 1974: Vint Cerf, Yogen Dalal a Carl Sunshine
- Zajišťuje
  - spolehlivost
  - ochranu proti zahlcení (různé algoritmy)
  - uspořádání paketů

# Mechanismy potvrzování

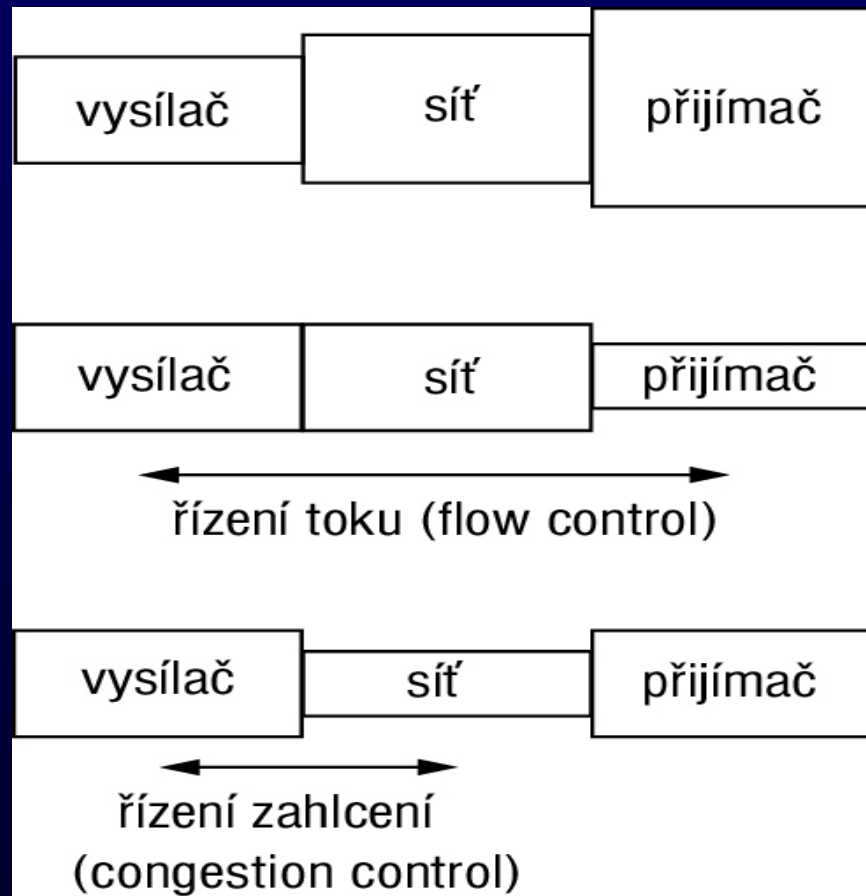
- kumulativní potvrzování
- piggybacking (duplexní protokol)
  - potvrzení je součástí zprávy jdoucí opačným směrem
- jedno potvrzení na 2 segmenty
  - je-li přenos dost rychlý (timeout 500ms)
- duplikované potvrzení
  - indikuje ztracený (poškozený) paket

# Zpracování malých zpráv

- Nagleův algoritmus
  1. Pokud proces  $p$  obdržel potvrzení pro všechna data dříve odeslaná  $q$ , potom proces  $p$  posílá zprávu ihned
  2. Pokud proces  $p$  neobdržel potvrzení pro všechna data dříve odeslaná  $q$ , potom proces  $p$  uloží zprávu pro pozdější odeslání
  3. Pokud velikost uložených zpráv přesáhne maximální velikost segmentu (MSS), tak se začne odesílat

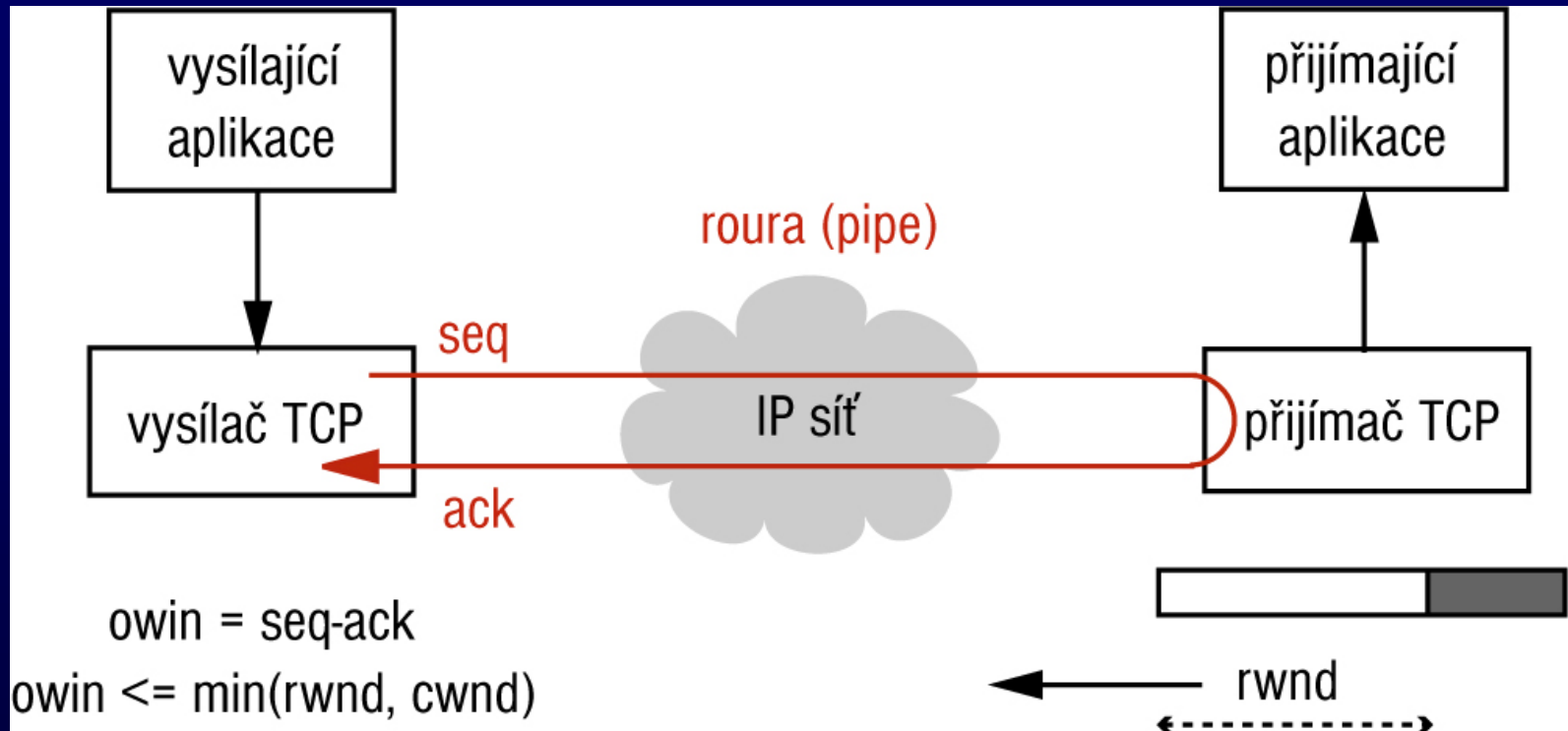
# Řízení toku

- řízení velikosti okna





# Okno TCP

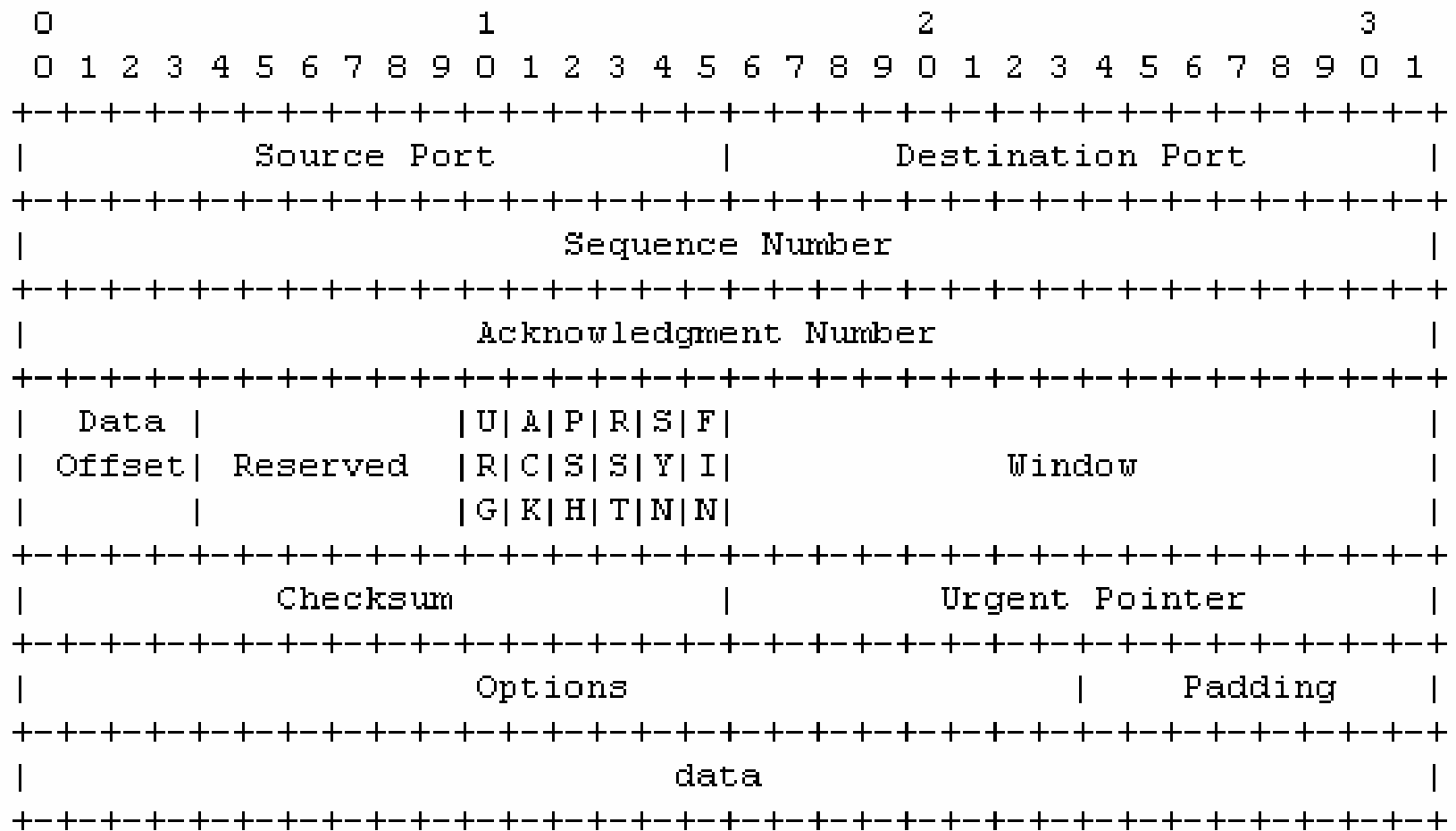


- outstanding window (ownd)
- receiver window (rwnd)
  - flow control, self-clocking
- congestion window (cwnd)

# Řízení zahlcení

- aproximativní
- detekce zahlcení pomocí výpadků
  - následná detekce
  - existují i preventivní mechanismy
  - závislost na informacích ze sítě
- congestion window (cwnd)
  - $bw = 8 * MSS * cwin / RTT$   
v případě že *owin* je limitováno *cwin*

# Hlavička TCP



# Ustavení spojení

- Entity (programy) A a B, sekvenční čísla SEQ(A) a SEQ(B)
- A: SYN, RAND(SEQ(A)) -> B
- B: SYN/ACK, RAND(SEQ(B)), SEQ(A)+1 -> A
- A: ACK, SEQ(B)+1 -> B
- => three-way handshake

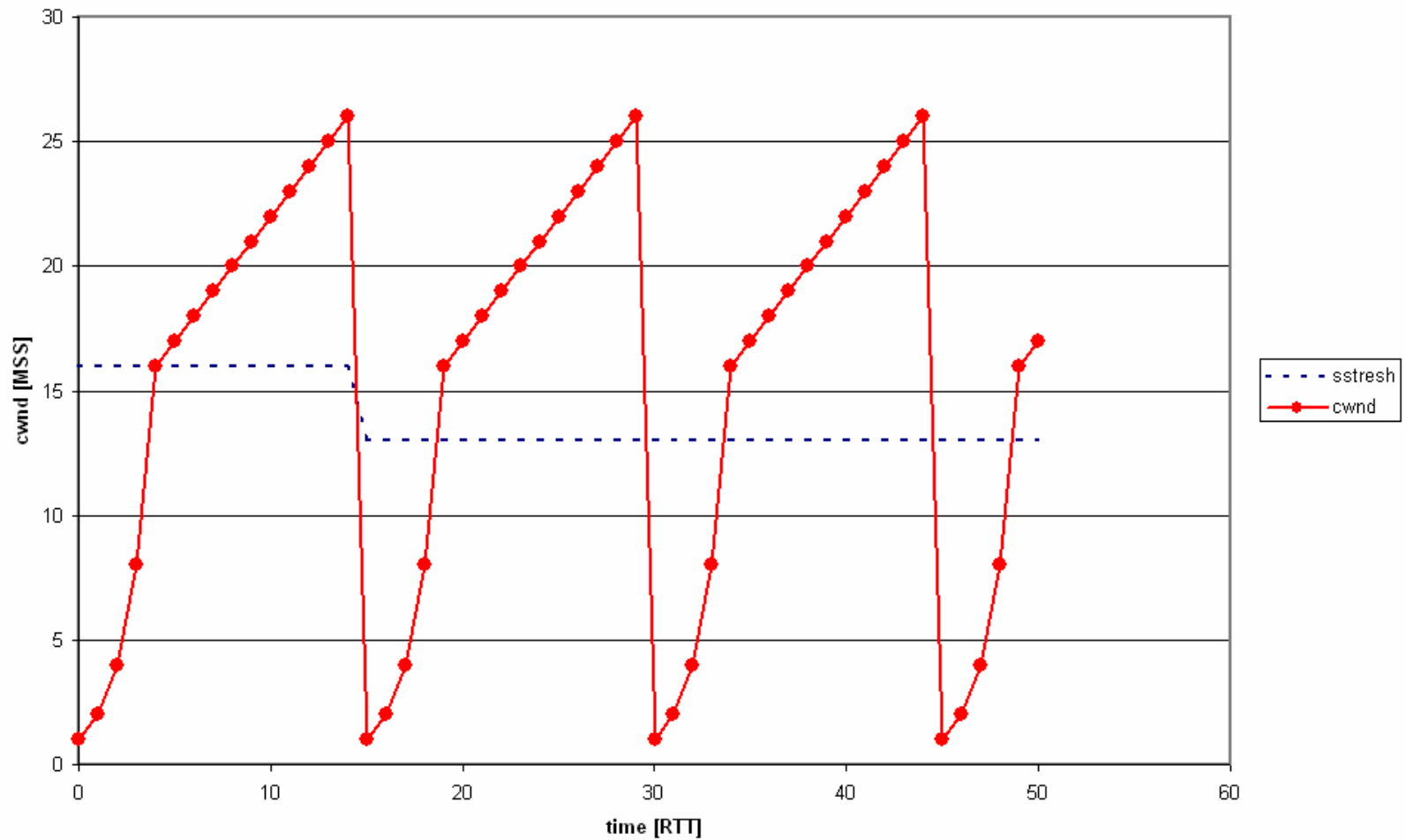
# Fáze Slow Start

- $cwnd += MSS$   
pro každý ACK
- exponenciální nárůst  $cwnd$
- limitováno  $ssthresh$
- přechod do congestion avoidance fáze

# Fáze Congestion Avoidance (Tahoe)

- AIMD (Tahoe)
  - additive increase, multiplicative decrease
  - $cwnd += 1 \text{ MSS}$   
během každého RTT bez výpadku
  - $cwnd = cwnd / 2$   
při výpadku
- Detekce výpadku
  - timeout
  - duplikované ACKy při výpadku
- $ssthresh = \max\{0.5 * cwnd, 2 * MSS\}$ 
  - návrat k slow start fázi

# Chování velikosti okna – Tahoe

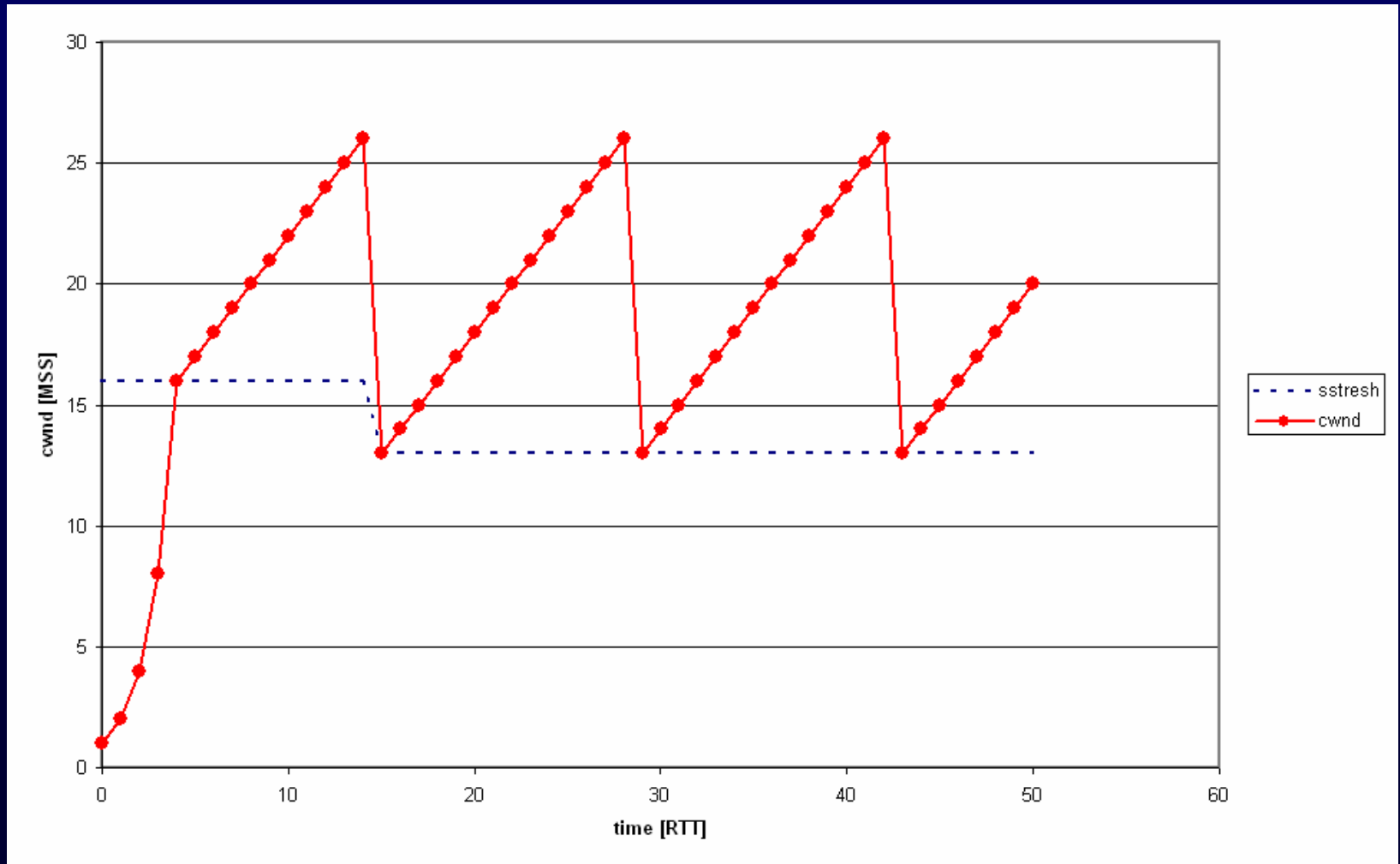


# Vylepšené chování – Reno

- fast retransmission
  - detekce výpadku 3 duplikovanými ACKy
- fast recovery
  - neprovádí se fáze slowstart
- $ssthresh = \max\{0.5 * cwnd, 2 * MSS\}$ 
  - (stejný jako TAHOE)
- $cwnd = ssthresh + 3 * MSS$



# Chování velikosti okna – Reno



# Reakce na zahlcení

- celé aktuální okénko (owin) – TCP Tahoe
  - klasický cumulative ACK
- jeden segment v rámci „Fast Retransmit” – TCP Reno
- více segmentů v rámci “Fast Retransmit” – TCP NewReno
- právě ztracené segmenty – TCP SACK
  - blokové potvrzení

# Alternativní přístup k řízení zahlcení – Vegas

- monitoruje RTT
- zahlcení detekuje pomocí prodlužování RTT
- na zahlcení reaguje lineárním poklesem *cwnd*

# TCP Response Function

- Vztah mezi rychlostí výpadků a propustností (owin/bw)
  - $owin \sim 1.2/\sqrt{p}$
  - $bw = 8 * MSS * owin / RTT$
  - $bw = (8 * MSS / RTT) * 1.2 / \sqrt{p}$
- p... packet loss rate [packet/s]

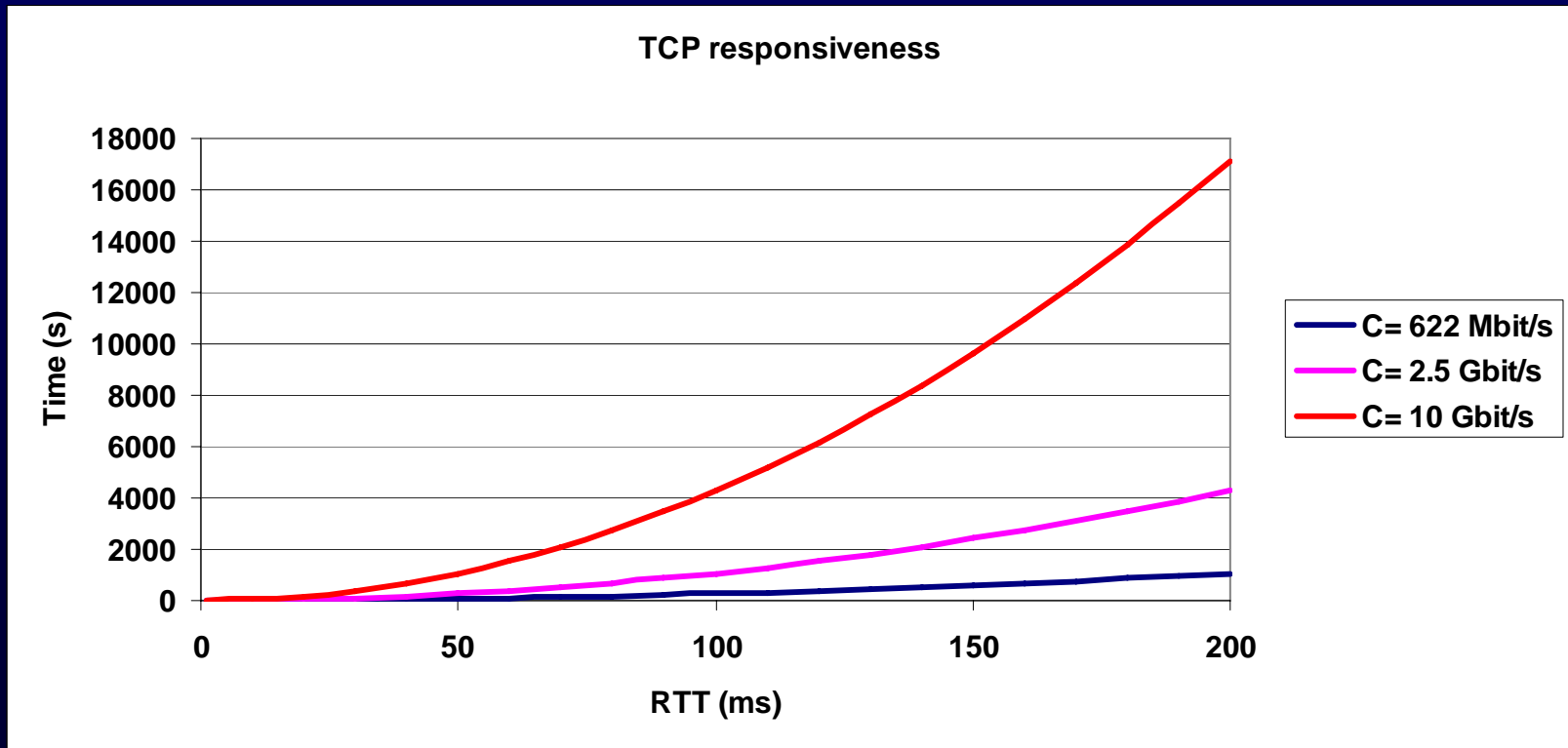
# Responsivnost TCP

- Schopnost návratu na plnou přenosovou rychlost po výpadku
- Přepokládejme výpadek v okamžiku, kdy  $cwnd = bw * RTT$

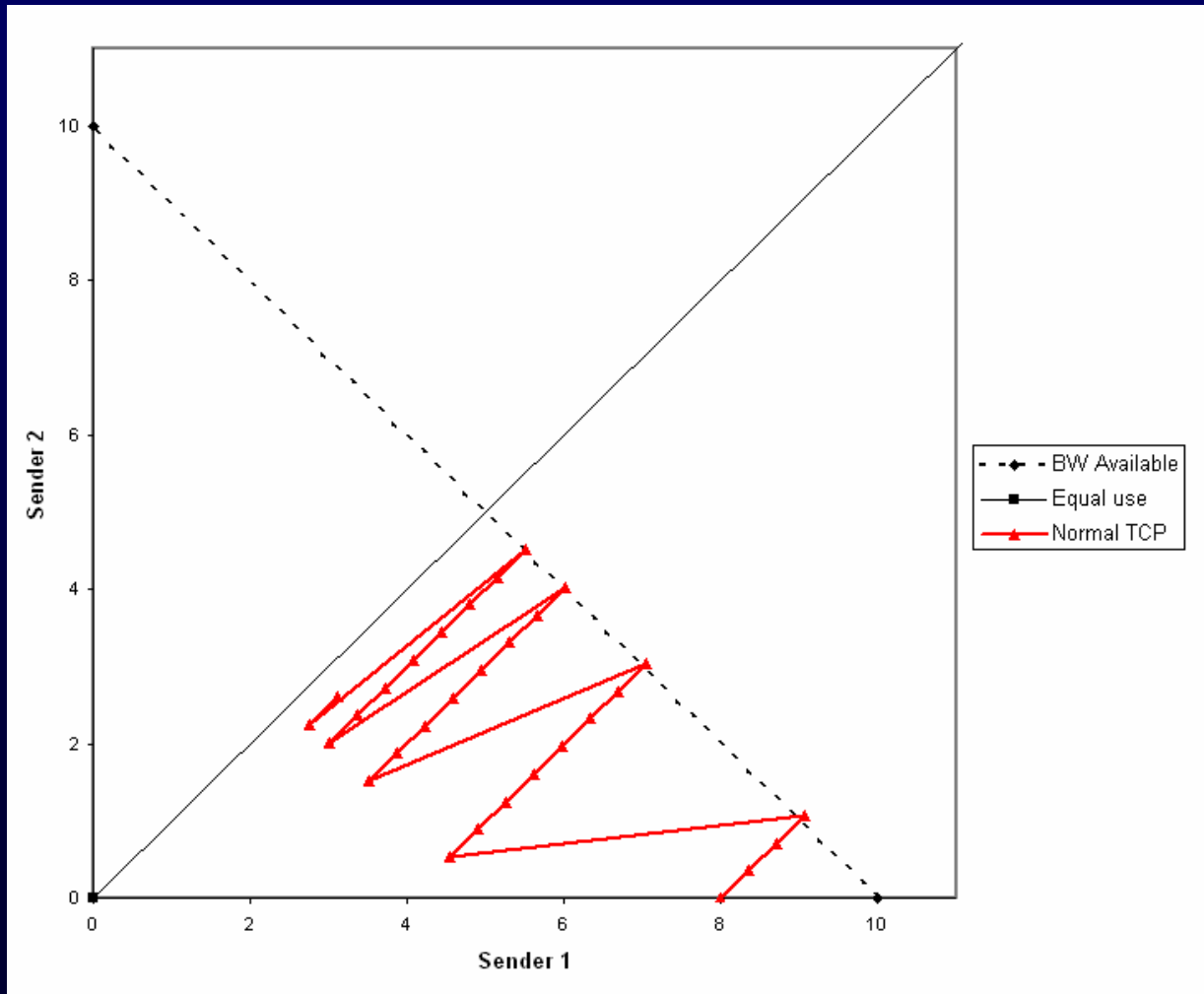
$$\rho = \frac{bw \cdot RTT^2}{2 \cdot MSS}$$

- závislost na RTT a MSS/MTU

# Responsivnost TCP (2)



# Férovost TCP



# Vylepšování implementace TCP

- checksum offloading
  - jak při vysílání, tak při příjmu
- zero copy
  - uživatelský prostor <-> jádro <-> karta
  - page flipping
  - Linux, FreeBSD, Solaris



Povídání o současném dění kolem  
zajištěných protokolů pro přenosovou  
vrstvu na PSaA II.