ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ
KATEDRA TELEKOMUNIKAČNÍ TECHNIKY

Pokročilé síťové technologie

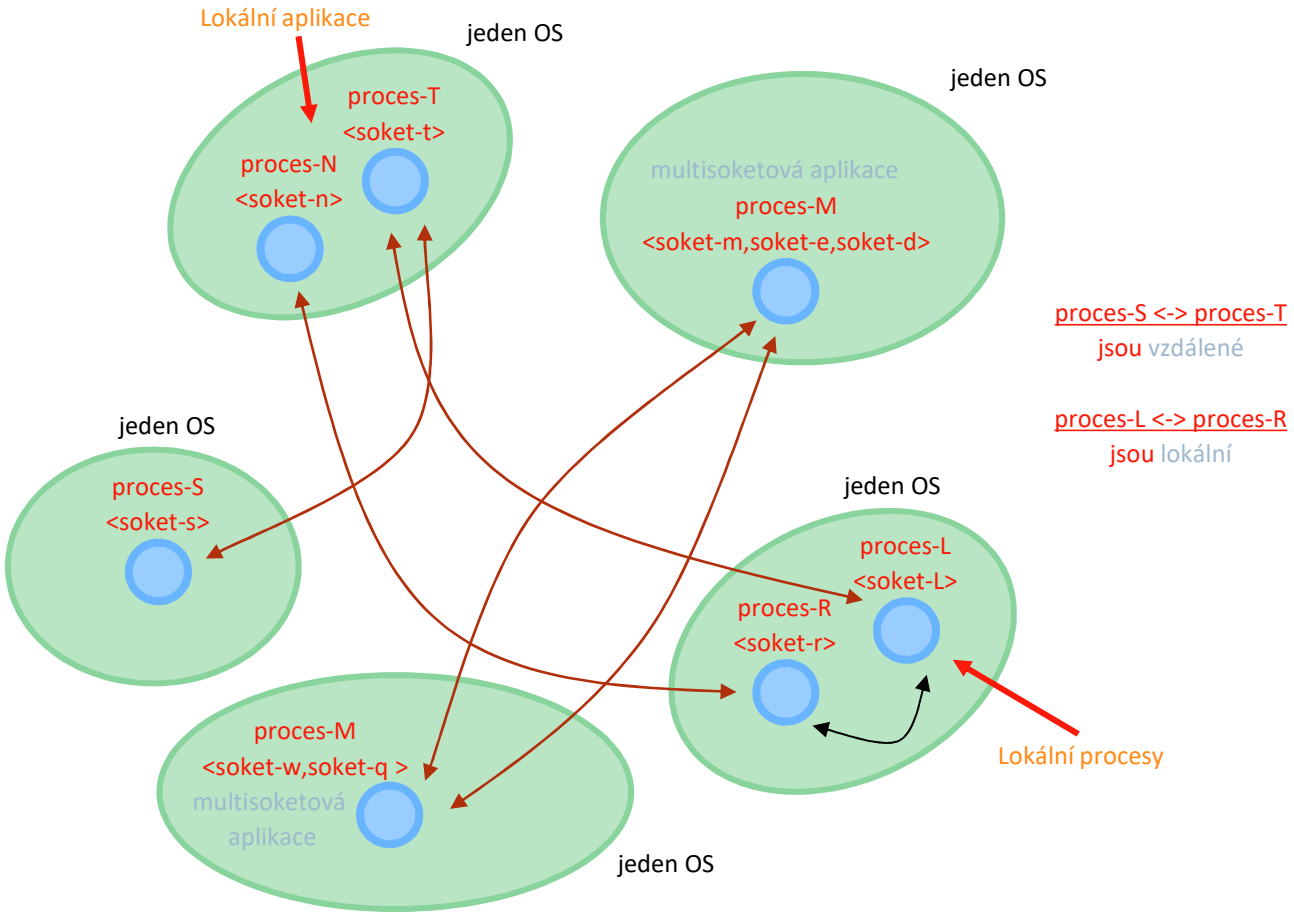
Transmission Control Protocol – TCP, Sockets

Doc. Ing. Leoš Boháč, Ph.D.

Procesy v architektuře - TCP/IP

- **síťová aplikace** v architektuře datových sítí **TCP/IP** většinou **integruje vždy tři nejvyšší vrstvy RM-OSI** modelu (aplikační, prezentační a relační)
- **procesy aplikace** mezi sebou **komunikují** prostřednictvím **speciálního rozhraní (API) – tzv. soketové rozhraní**
- všechny **procesy** jsou **součástí aplikačního prostoru**
- aby mezi sebou mohly procesy komunikovat, musí být navzájem jednoznačně identifikovatelné
- **každý proces je** v architektuře TCP/IP **jednoznačně identifikován** jedním nebo více **soketovými identifikátorů <sok1, sok2, ..>,** jejichž přesná syntaxe bude vysvětlena později v souvislosti s transportními protokoly UDP, popř. TCP
- jeden soketový identifikátor <sok> jednoznačně určuje obousměrný datový okruh (pár kanálů) mezi komunikujícími procesy, který je typicky schopen přenášet data ve formě datových bloků různé délky a s různou garancí kvality přenosu (chybovost, zpoždění, apod.)
- **přenos zpráv** mezi procesy apliakce může být **jednosměrný (simplexní)** nebo **obousměrný (duplexní)**, nicméně **model přenosu** na transportní vrstvě **TCP nebo UDP** uvažuje vždy **obousměrný přenos**
- z pohledu komunikace mezi procesy je **lhostejné**, jestli jsou **lokální nebo vzdálené** – transportní protokoly zajistí transparentnost
- procesy komunikující v rámci jednoho OS (typicky jeden fyzický stroj, např. PC, laptop nebo skupina strojů, které jsou mezi sebou propojené jen lokální sběrnicí na krátkou vzdálenost do cca 10 m) se nazývají **lokální, (local) lokální procesy se vyznačují použitím speciálního soketového identifikátoru <soklocal>**
- procesy, které si vyměňují informace přes datovou síť v rámci LAN, MAN nebo WAN se nazývají **vzdálené (remote)**

Aplikační prostor – množina komunikujících procesů

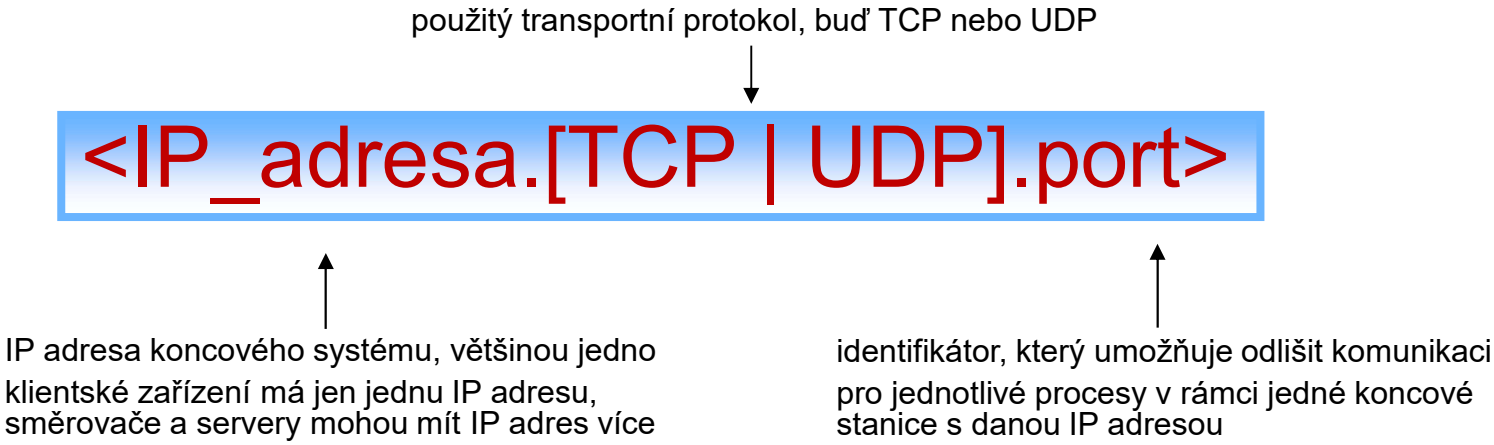


Model komunikace mezi procesy

- historicky prvním modelem komunikace mezi procesy je model založený na vztahu **klient/server** (Client/Server)
  - v tomto případě se předpokládá, že serverový proces aplikace disponuje určitými technickými a informačními prostředky, kterými nemůže za normálních okolností disponovat klientská aplikace
  - model klient/server je vhodný pro systémy, kde se např. na serverové části sdílí určité technické prostředky (tiskárny, kreslící zařízení, faxy, atd.) nebo kde se nachází jednotný centralizovaný zdroj informací (např. databáze) nebo kde je požadována centralizovaná správa dat či prostředků
  - je to dnes nejčastěji používaný model – historicky je daný tím, že dříve používaná klientská zařízení technicky nemohla mít k dispozici velké úložné kapacity, popř. nebylo možné z finančních důvodů ke každé stanici připojit kdysi drahé zařízení, jakým byla např. tiskárna či plotter
  - komunikace je v tomto případě založena na principu dotaz/odpověď, v odpovědi může být obsaženo větší množství dat, než v dotazu (např. WEB, FTP, apod.) – objem přenosu dat je tedy často asymetrický
  - sdílení dat a prostředků má centralizovaný charakter
- model založený na systému komunikace **rovný s rovným** (Peer-to-Peer)
  - v tomto případě jsou koncová zařízení z hlediska možnosti vybavení, úložných kapacit nebo připojených technických prostředků na tom identicky (jsou si sobě rovna – od toho pojem „peer“)
  - každá aplikace v tomto modelu předává jiné aplikaci průměrně stejný objem dat
  - charakter přenosu je většinou symetrický (Napster, Kaza, Skype, Direct Connect, BitTorrent)
  - sdílení dat a prostředků má primárně distribuovaný charakter

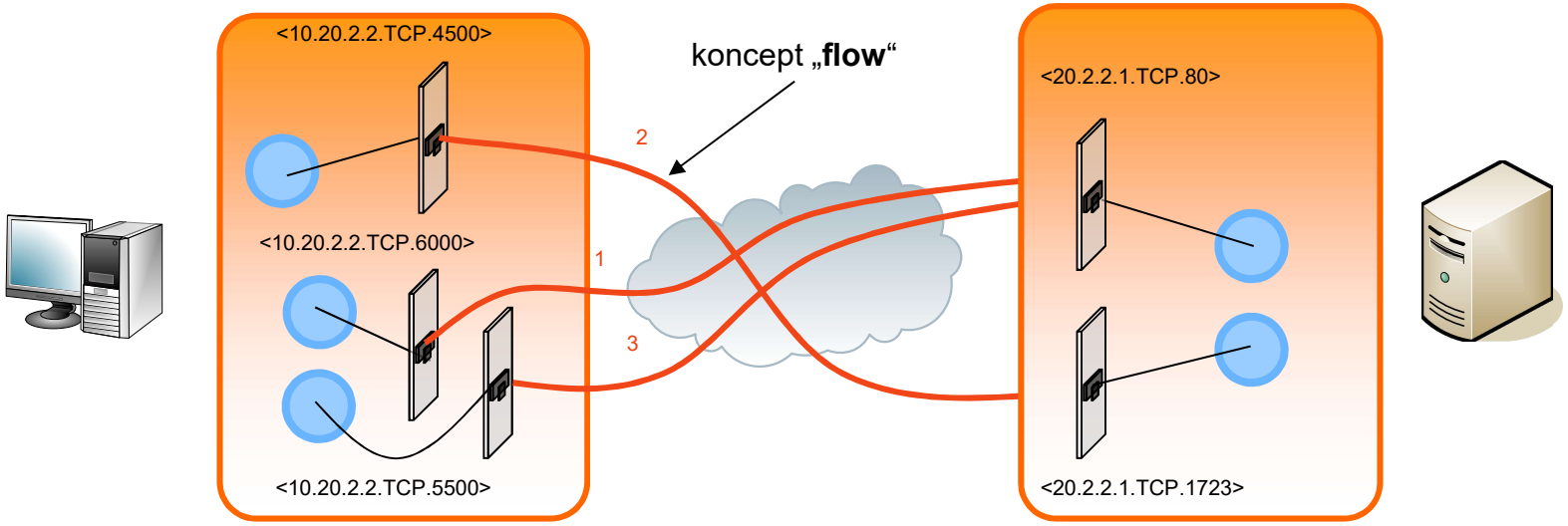
Identifikace koncového bodu komunikace v TCP/IP síti

- každý koncový bod komunikace v rámci TCP/IP sítě je jednoznačně určený svou hodnotou soketu



<147.32.192.23.TCP.80>
<200.2.3.4.UDP.123>
<65.2.34.2.TCP.1723>
<65.2.34.2.TCP.53>
<2002:2020::01.53>

Použití a párování „sockets“



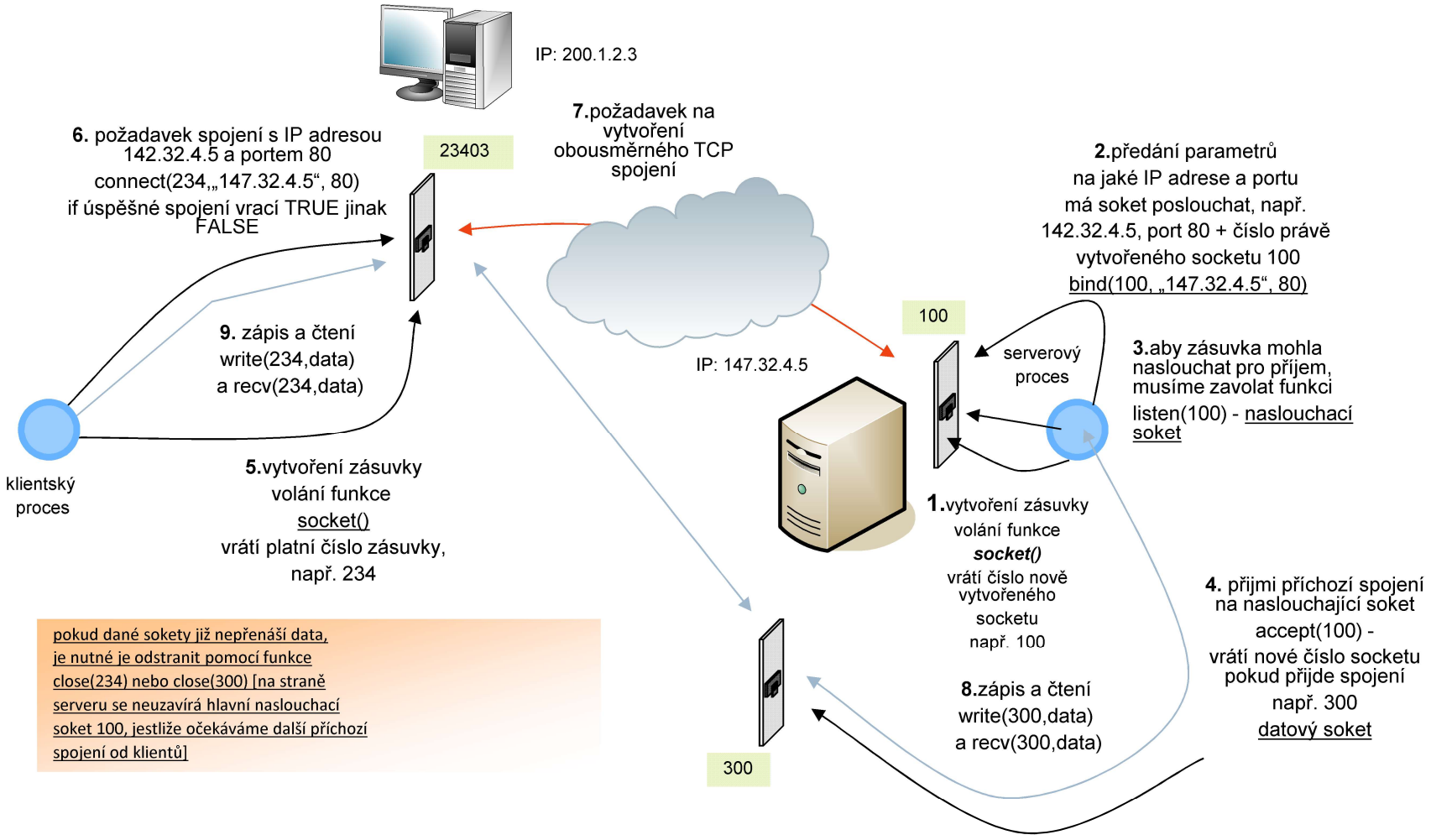
komunikační okruh je jednoznačný, pokud je pár zdrojového a cílového soketu jednoznačný:

- kanál č.1 je určený párem soketů <10.20.2.2.TCP.6000, 20.2.2.1.TCP.80>
- kanál č.2 je určený párem soketů <10.20.2.2.TCP.4500, 20.2.2.1.TCP.1723>
- kanál č.3 je určený párem soketů <10.20.2.2.TCP.5000, 20.2.2.1.TCP.80>

Procesy a aplikace v TCP/IP

- aplikační vrstva v rámci RM-OSI je poněkud abstraktní pojem
- aplikace obecně může zahrnovat několik běžících procesů, jak lokálních, tak i vzdálených
- pro zjednodušení situace nyní předpokládejme, že jeden proces je roven jedné aplikaci v modelu TCP/IP
- u většiny moderních OS se komunikace v prostředí TCP/IP sítí odehrává prostřednictvím programového rozhraní zvané síťové zásuvky (sockets) (převzaté z UNIXU v Berkley BSD)
- pro spolehlivý přenos dat mezi aplikacemi (procesy) se používá funkce TCP transportního protokolu, u něhož je typicky nutné (ale ne nezbytné), aby jedna strana aktivovala spojení (většinou klient) a druhá strana na spojení čekala (většinou server strana)
- programové rozhraní zásuvek (sockets) tedy také vychází z modelu komunikace klient/server
- po aktivace serverové části zásuvky ona vyčkává na příchozí požadavky spojení od klientů !!
- klienti naopak typicky spojení zahajují !!
- proces, který používá danou zásuvku je s ní pevně spjatý až do doby jejího odstranění
- každá nově vytvořená zásuvka je v OS jednoznačně identifikována celých číslem, tzv. „handlem“ (referečním číslem)
- v OS nemohou být za normálních okolností dvě zásuvky stejného typu se stejným „handlem“(referečním číslem)
- pokud komunikační kanál reprezentovaný danou zásuvkou není už zapotřebí, zásuvku lze zrušit, čím její objekt zanikne a taktéž i s ní existující spojení

Princip spojení pomocí soketů



Ilustrační příklad v PHP ….

SERVER (jen ilustrační příklad soketové komunikace !!)

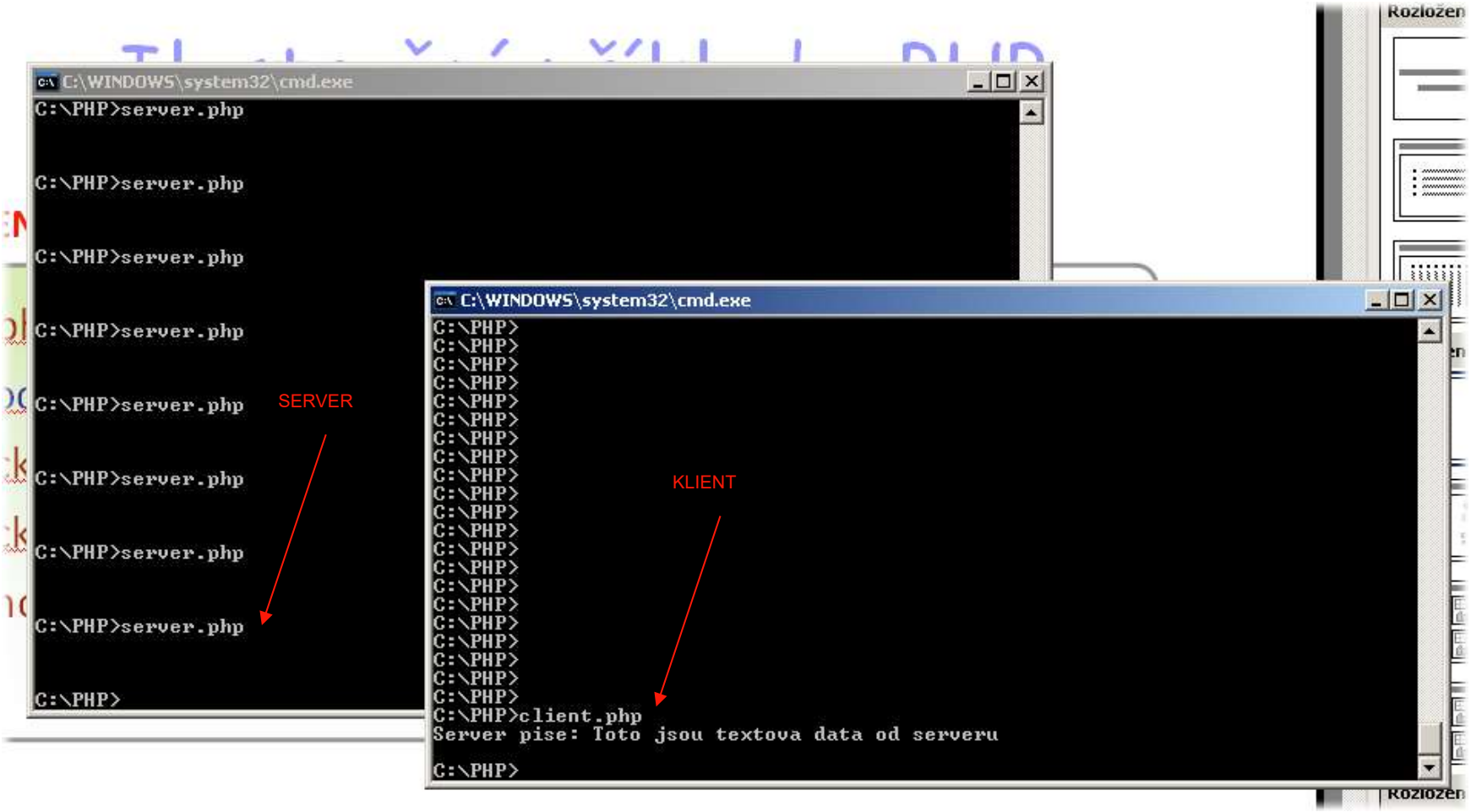
```
<?php
$listen_socket=socket_create(AF_INET,SOCK_STREAM,SOL_TCP);
socket_bind($listen_socket,'127.0.0.1',5000);
socket_listen($listen_socket);
$data_socket = socket_accept($listen_socket);
$data="Toto jsou textova data od serveru";
socket_write($data_socket,$data);
socket_close($data_socket);
?>
```

Ilustrační příklad v PHP ….

KLIENT (jen ilustrační příklad soketové komunikace !!)

```
<?php
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
socket_connect($socket,"127.0.0.1", 5000);
socket_recv($socket,$buff, 8000, 0);
echo "Server pise: ".$buff;
?>
```

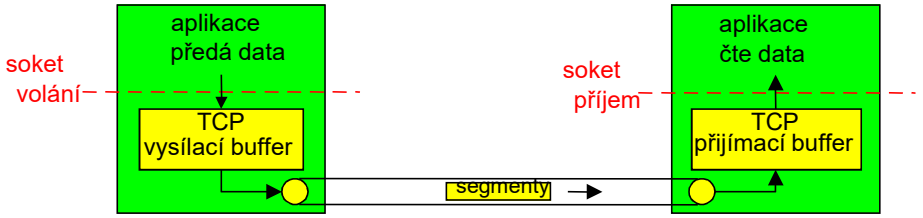
Ilustrační příklad v PHP ….



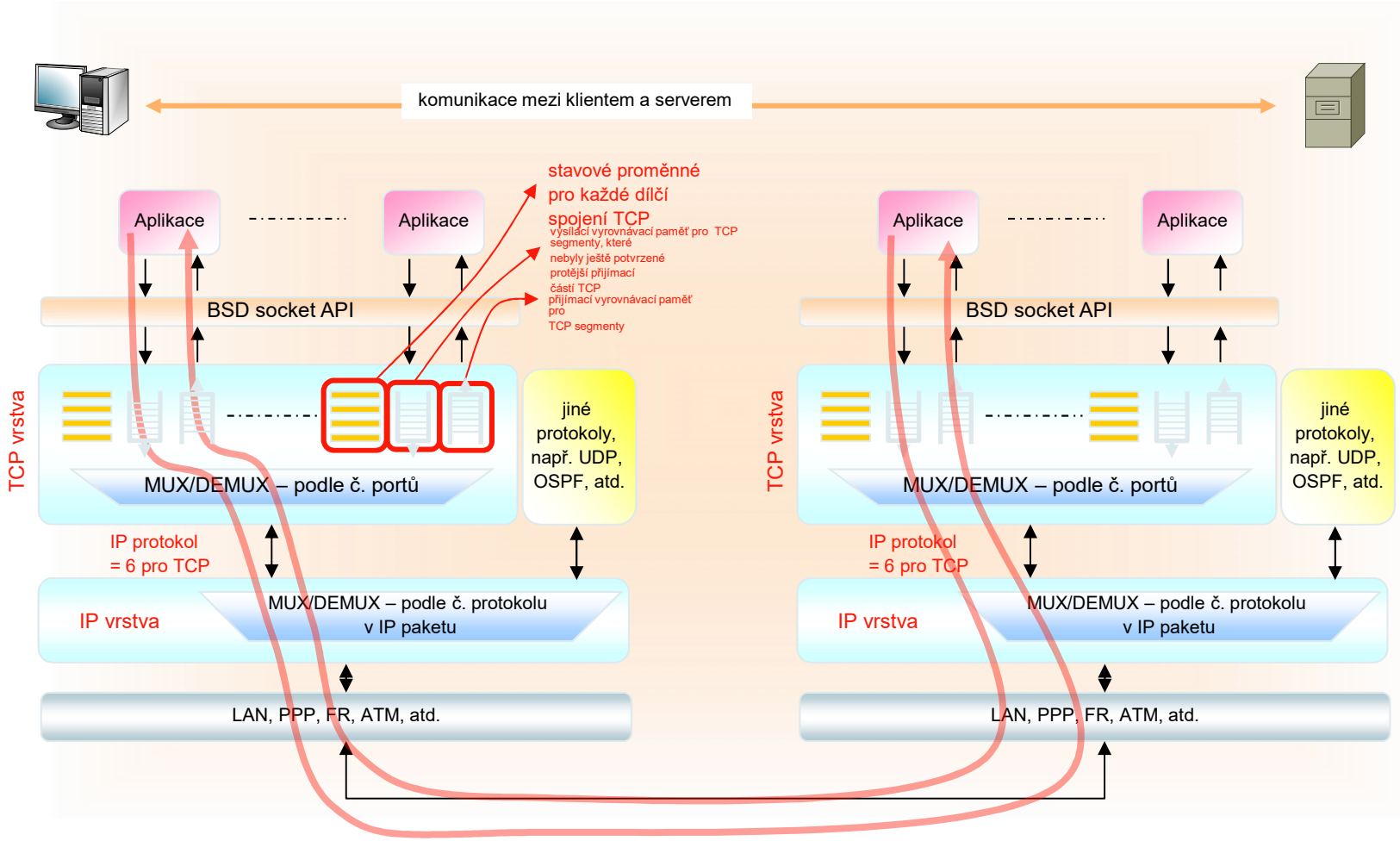
Transmission Control Protocol - TCP

- **TCP „roadmap“ v RFC4614**
- RFCs: 793, 1122, 1323, 2018, 2581, atd
- dvoubodové spojení:
  - jeden vysílač, jeden přijímač
- použití metody klouzavého okna
- používá vysílací a přijímací paměť (buffered)

- spojově orientované řešení přenosu dat:
  - výměna řídících informací prostřednictvím tří zpráv (-> SYN, <- SYN, -> SYN, ACK)
  - přad vlastním přenosem dat je nutné inicializovat vysílač a přijímač – sekvenční čísla
- zajišťuje plně duplexní přenos dat:
  - v jednom spojení přenos v obou směrech
  - MSS: maximální velikost segmentu (maximum segment size)
- zajišťuje řízení toku (flow control) a řízení přetížení (congestion control)



Model komunikace TCP



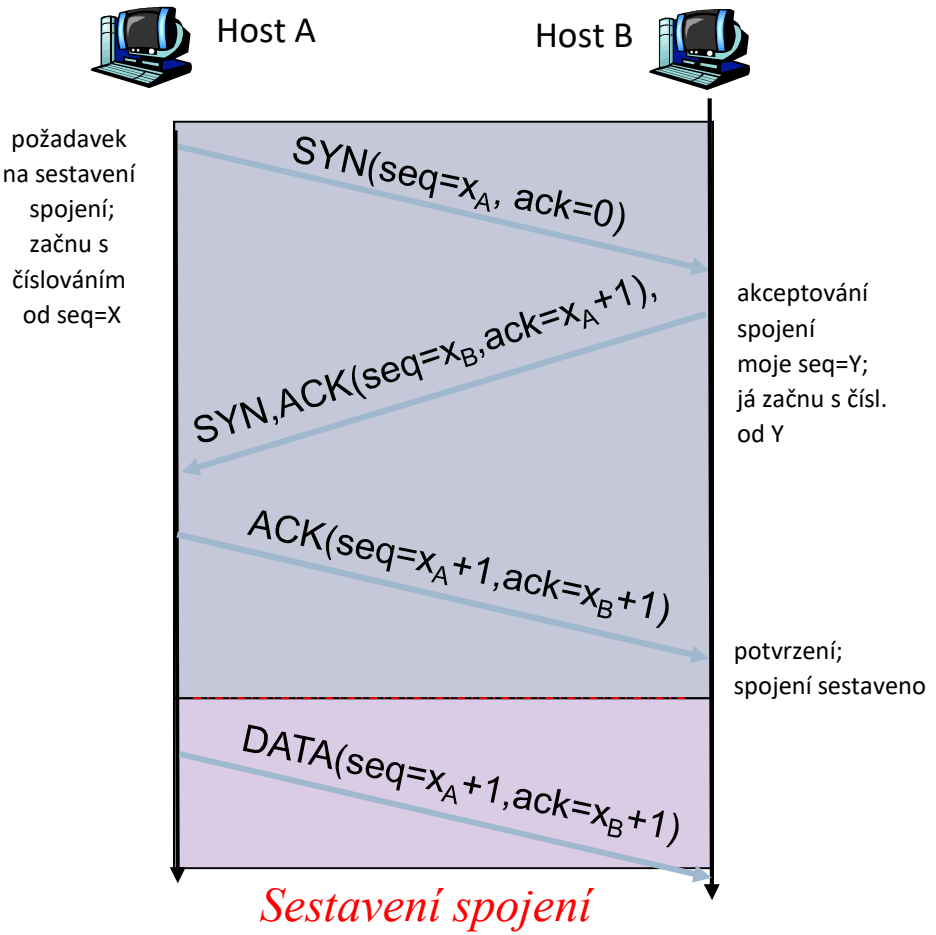
Rozdělení funkcí u TCP - vysílání

- rozděluje datové bloky do **segmentů** a každý opatří **pořadovým číslem** (Sequence Number)
- vysílá připravené segmenty do sítě pomocí IP datagramů až do vyčerpání **vysílacího okna** (Send window)
- pokud nepřijde potvrzení o příjmu segmentu po uplynutí **RTO (Retransmission TimeOut)** časovače, vyšle daný segment znovu
- přijímá průběžně jednotlivá **potvrzení** od přijímače (**ACK - acknowledgement**)
- s každým **ACK** posune okno o velikost potvrzeného segmentu
- při příjmu **tři stejných ACK** pro daný segment vyšle chybějící segment bez čekání na vypršení RTO časovače – (fast retransmit)
- zajišťuje regulaci rychlosti vysílaných segmentů do sítě s cílem zabránit vzniku přetížení sítě (congestion control, congestion window)

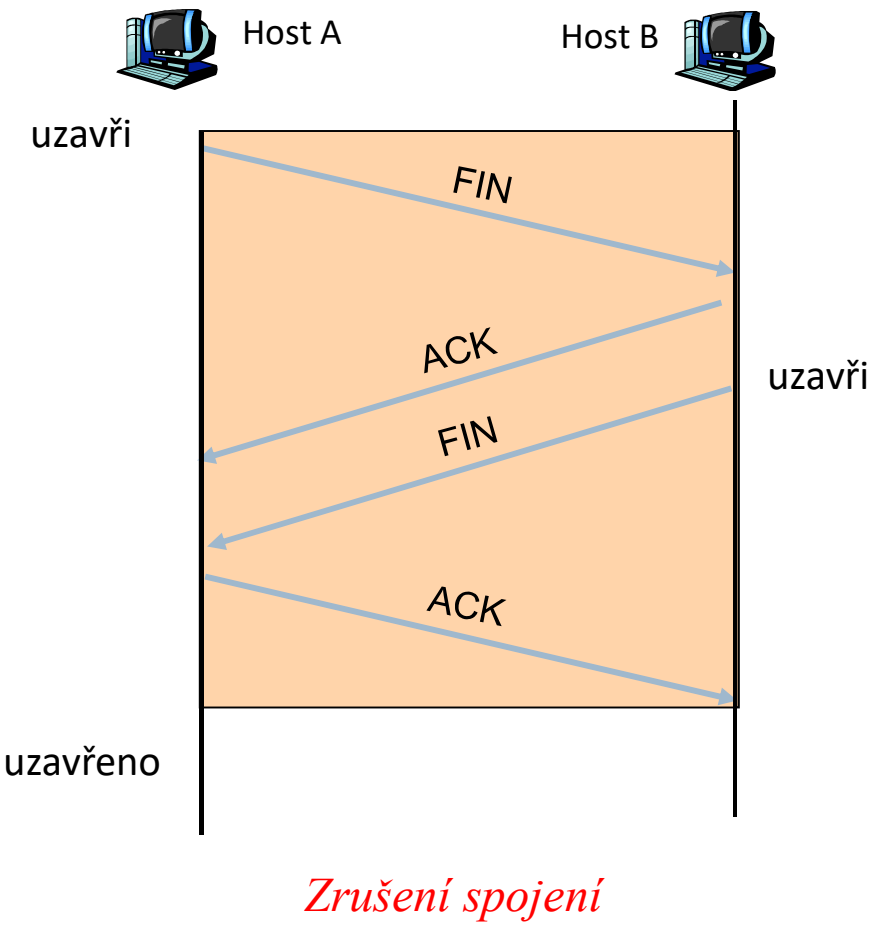
Rozdělení funkcí u TCP - příjem

- přijímá segmenty a sestavuje je ve vyrovnávací paměti do správného pořadí
- pro **každý, max. každý druhý segment** (**Delayed Acknowledgement**), posílá potvrzení, pokud byly doručeny segmenty mimo správné pořadí, posílá okamžitě ACK s číslem segmentu, který tvoří ve vyrovnávací paměti spodní část mezery (viz dále)
- zajišťuje **zpožděné potvrzování**, tj. při příjmu segmentu ve správném pořadí, ještě čeká max. **500 ms** jestli nepřijde následující segment; pokud ano, potom vyšle jedno potvrzení pro oba segmenty – **vícenásobné potvrzení** (**Cumulative Acknowledgement**)
- umožňuje **řízení toku** nastavením okna pro vysílač (přenášeno od přijímače k vysílači v TCP poli **okno** - Window)

Sestavení a zrušení spojení u TCP

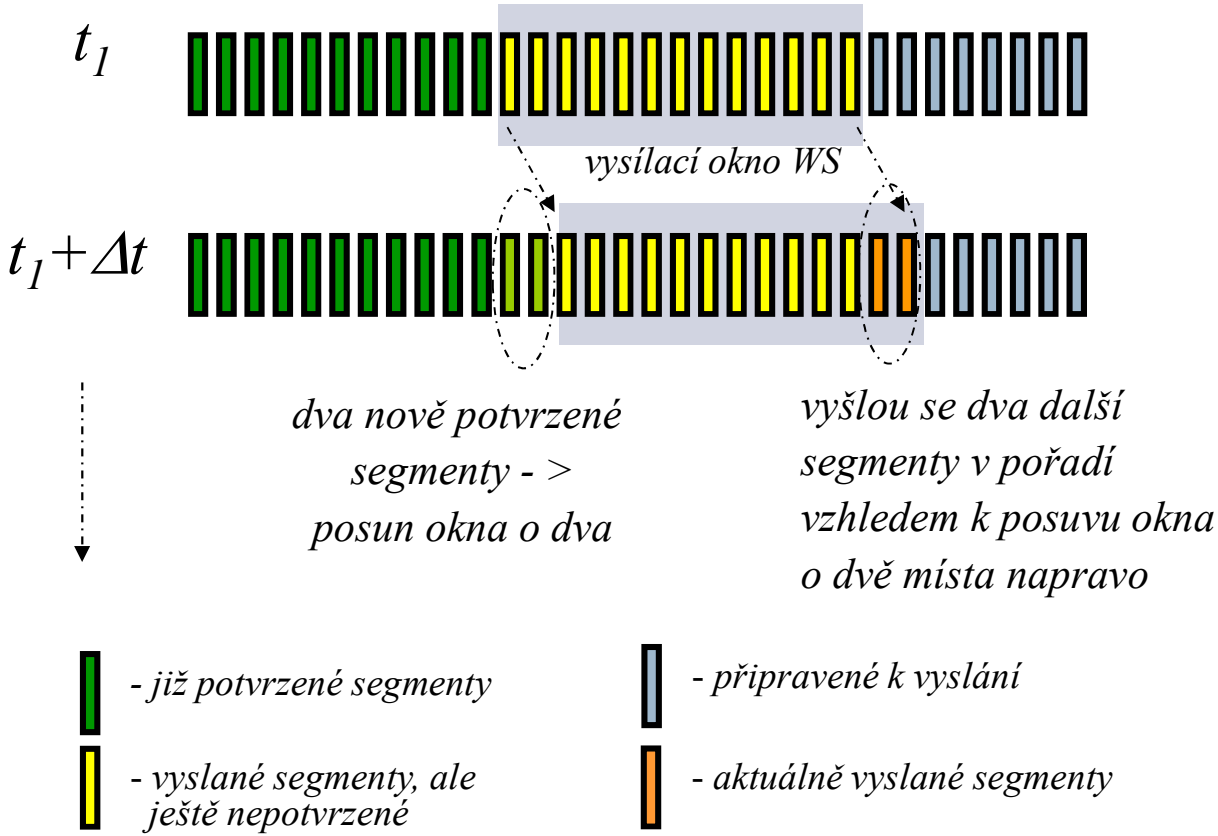


Sestavení spojení



Zrušení spojení

Metoda klouzavého TCP okna



Maximální propustnost TCP spojení

- maximální propustnost TCP spojení je dána vztahem

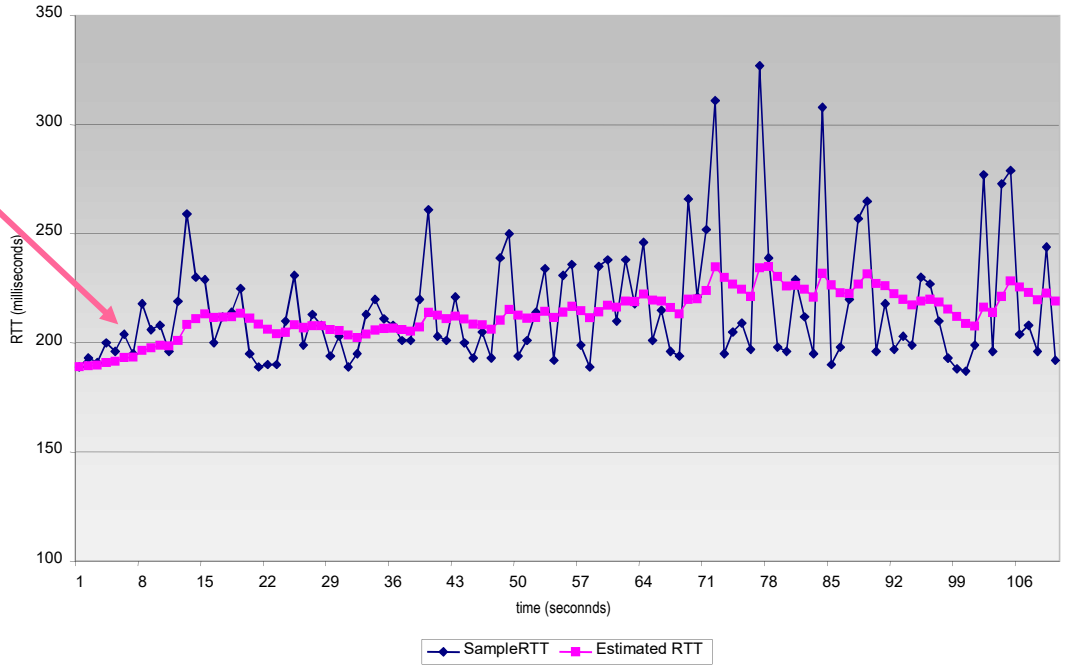
$$\text{propustnost} = \frac{8*WS}{RTT} \quad [\text{bit/s}]$$

- kde: WS – velikost okna na straně vysílače a RTT – „round trip time“ - čas, který uplyne od vyslání paketu k jeho příjmu a přijetí potvrzení

Závislost RTT v čase

- časový interval, který uplyne od vyslání segmentu do příjmu jeho potvrzení ACK se **nazývá časový interval odezvy RTT** (Round Trip Time)
- *RTT není v čase konstantní, ale mění se v závislosti na aktuálním stavu zatížení cesty v paketové síti*
- časovač **opětovného vyslání segmentu - RTO** (Round Trip TimeOut) se musí automaticky nastavovat podle aktuálních podmínek zatížení v datové TCP/IP síti

Odhad_RTT



How to set the timer

- **Retransmission Timer:**
  - The setting of the retransmission timer is crucial for good performance of TCP
  - Timeout value too small → results in unnecessary retransmissions
  - Timeout value too large → long waiting time before a retransmission can be issued
  - A problem is that the delays in the network are not fixed
  - Therefore, the retransmission timers must be adaptive

Časovač opětovného vysílání – RTO (Retransmission Time Out)

- pokud je časovač **nastaven na krátký čas**, bude docházet k častému a zbytečnému opakování vysílání TCP segmentů, které přijímač již správně přijal – zbytečné plýtvání kapacity sítě
- pokud bude časovač **příliš dlouhý**, bude **klesat efektivní přenosová rychlost TCP spojení dané relace a doba zpoždění přenosu dat**
- **RTT je náhodná veličina** s konkrétní střední hodnotou a rozptylem
- nastavení časovače je nutné řešit dynamicky a vzít v úvahu rozptyl
- RTT se měří průběžně a poslední hodnota změřené hodnoty se ukládá do proměnné **aktuální RTT**
- po opětovném vyslání segmentu se zvětší časovač opětovného vyslání pro tento segment na dvojnásobek – exponenciální „**backoff**“

$$Odhad\_RTO = \alpha * Odhad\_RTO + (1-\alpha) * Aktuální\_RTT$$

- **pro výpočet se používá exponenciálně vážený klouzavý průměr**
- vliv předchozích hodnot na aktuální RTT exponenciálně klesá
- typicky: $\alpha = 0{,}8 - 0{,}9$

Nastavení časovače opětovného vysílání

- prosté nastavení časovače RTO na střední hodnotu RTT by vedlo k velkému množství opakovaných segmentů -> proto je nutné přičíst ještě rezervu DevRTT

$$DevRTT = (1-\beta)*DevRTT + \beta*|Aktuální\_RTT-Odhad\_RTTO|$$

$$(typicky: \beta = 0.25)$$

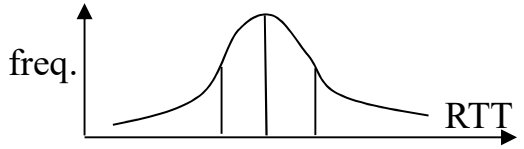
↓

časovač se tedy dynamicky nastavuje podle rovnice:

počáteční hodnota RTO == 3 s
maximální konečná hodnota RTO == 60 s

minimální hodnota platná pro RTO
v **RFC 2988** je **1 s** pro potlačení
nelegetimních retransmisí
a také původní malou granularitou časovačů
v OS
- toto může být problém
- není dnes zcela pravda

↓



$$RTO = max(1s, Odhad\_RTO + 4*DevRTT)$$

Retransmission Timer

- TCP sender **maintains one retransmission timer for each connection**
- The timer is started when
  a. Every time a packet containing data is sent (including a retransmission), if the timer is not running, start it running so that it will expire after RTO seconds (for the current value of RTO)
  b. When all outstanding data has been acknowledged, turn off the retransmission timer,
  c. When a segment is retransmitted

- When the retransmission timer expires, do the following:
  a. Retransmit the earliest segment that has not been acknowledged by the TCP receiver.
  b. The host MUST set RTO <- RTO * 2 ("back off the timer"). The maximum 60 s may be used to provide an upper bound to this doubling operation.
  c. Start the retransmission timer, such that it expires after RTO seconds (for the value of RTO after the doubling operation outlined in b)

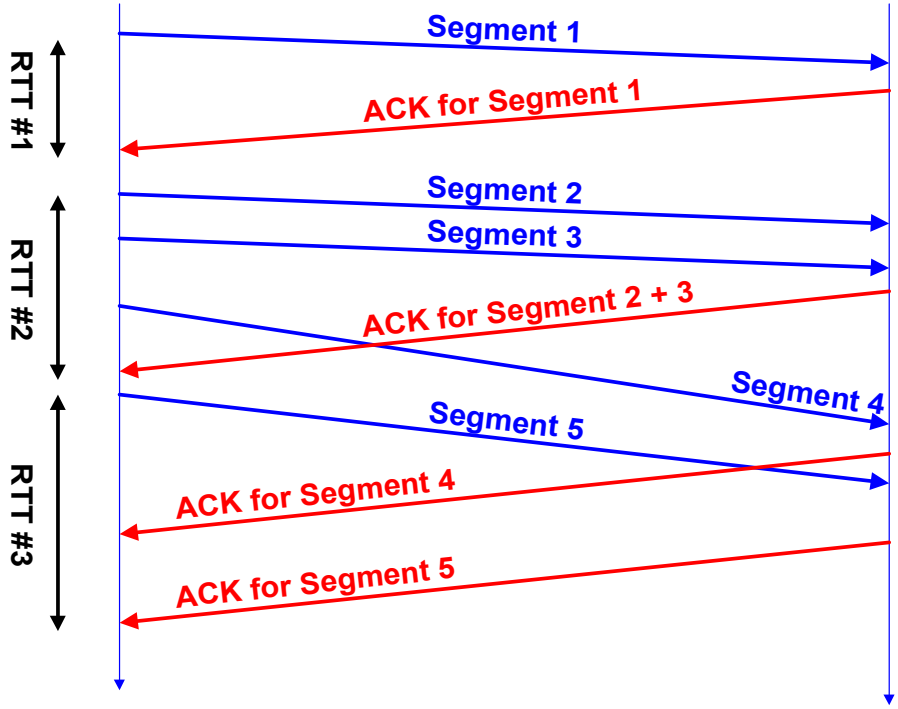
Setting the value of RTO:

- The RTO value is set based on round-trip time (RTT) measurements that each TCP performs

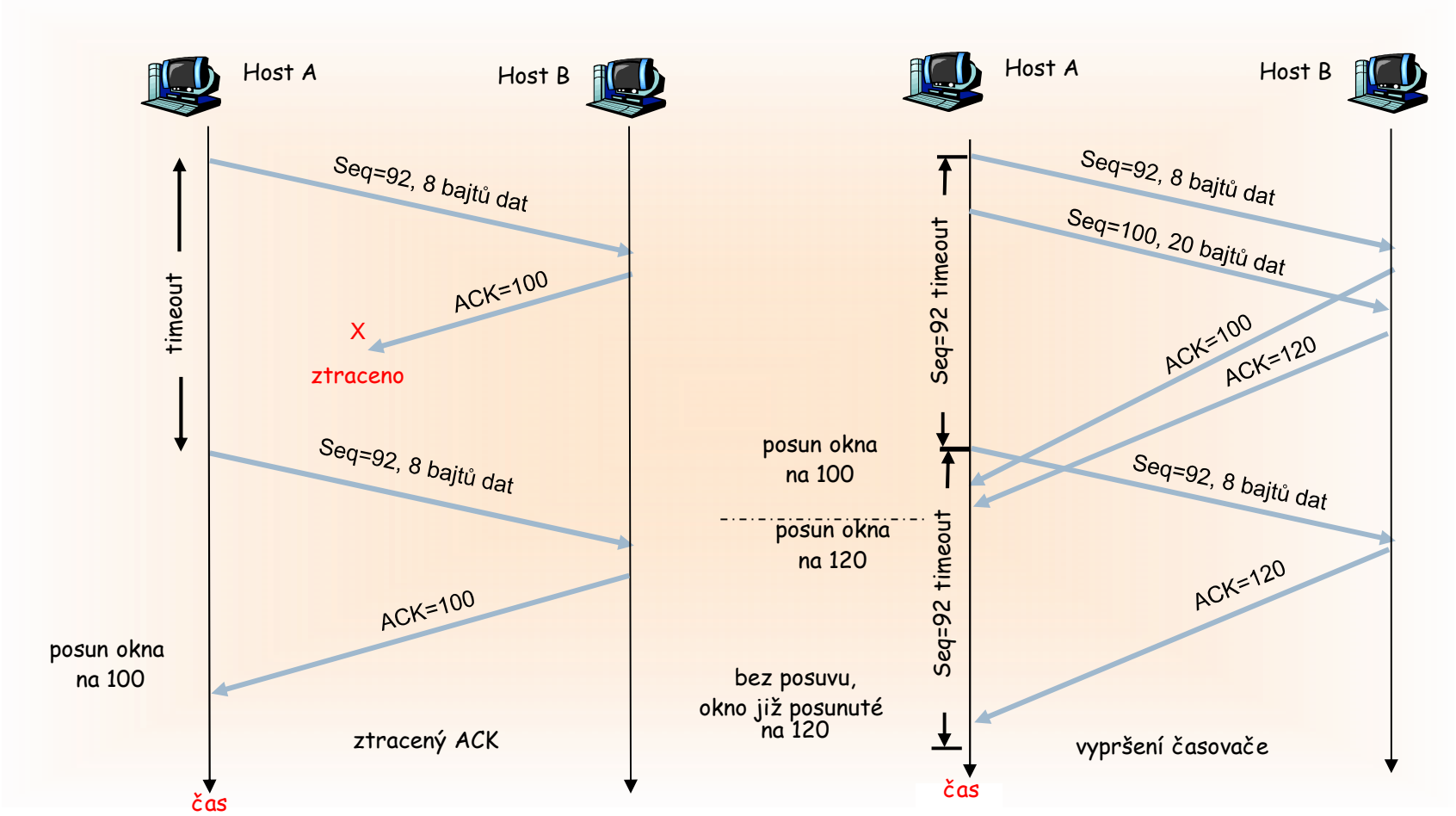
Each TCP connection measures the time difference between the transmission of a segment and the receipt of the corresponding ACK

There is only one measurement ongoing at any time (i.e., measurements do not overlap)

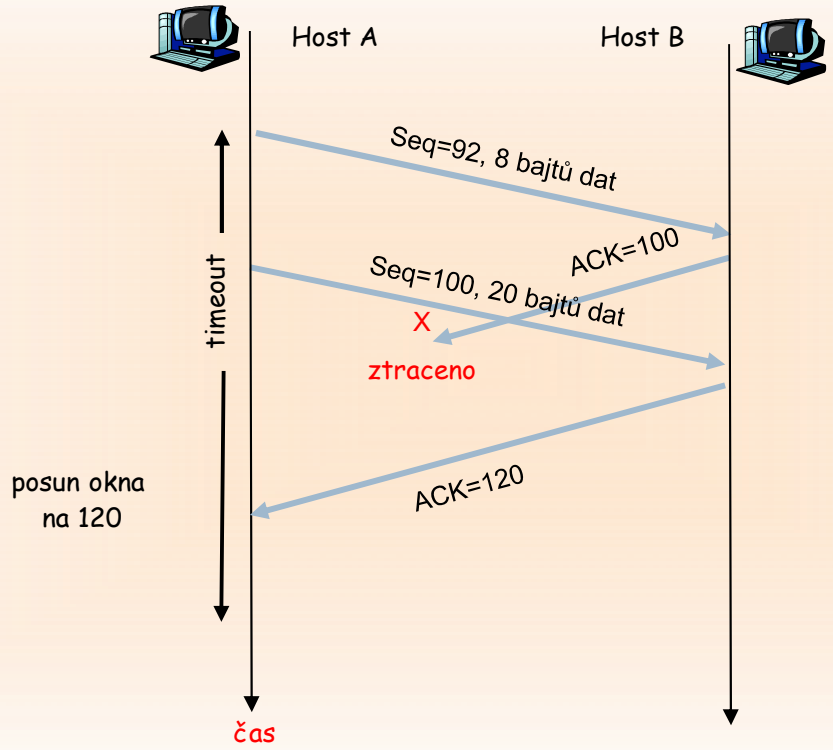
Figure on the right shows three RTT measurements



Opakované vysílání segmentu



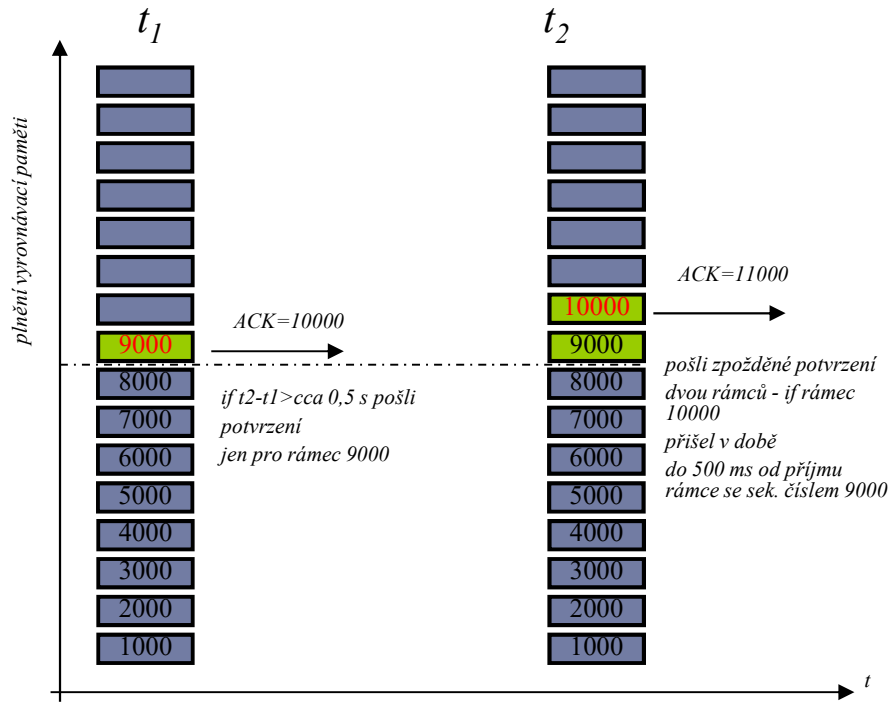
Vícenásobné potvrzení



hromadné potvrzení přijatých segmentů

Generování potvrzení u TCP

1. maximálně každý druhý bezchybně přijatý segment musí být potvrzen
2. po uplynutí časovače zpožděného potvrzení (delayed ACK) – max. 500 ms
3. při každém příchodu nového segmentu mimo pořadí, kdy existuje v přijaté sekvencí rámců prázdné místo („díra“)



Generování potvrzení u TCP

1. maximálně každý druhý bez chyb přijatý segment musí být potvrzen
2. po uplynutí časovače zpožděného potvrzení (delayed ACK) – max. 500 ms
3. při každém příchodu nového segmentu mimo pořadí, kdy existuje v přijaté sekvencí rámců prázdné místo („díra“)