

# Transportní protokoly

PDS (Přenos dat, počítačové sítě a protokoly)

Vladimír Veselý, [veselyv@fit.vutbr.cz](mailto:veselyv@fit.vutbr.cz)

# Agenda

- Paketové chyby a jejich příčiny
- Detekce chyb
- Korekce chyb
- Příklady transportních protokolů

# Úvod

# Motivace

## ■ Bitové chyby

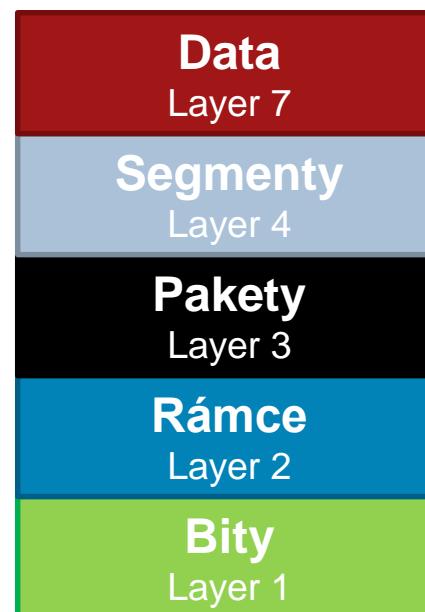
- *Chyby na úrovni „zpracování“ informace uzlem v síti*
- Zabýváme se jimi na L2
- Řešením je obvykle **redundance** – bezpečnostní kódy (Hamming, CRC, konvoluční kódování)

## ■ Paketové chyby

- *Chyby na úrovni „přenosu“ informace mezi dvěma uzly v síti*
- Zabýváme se jimi na L4
- Řešením je obvykle **znovuzaslání**

# Taxonomie

- **Datový tok** (aka dataflow/stream)  $\stackrel{\text{def}}{=}$  posloupnost segmentů procházející stejnými uzly sítě mající stejné vlastnosti (IP adresy, porty, ToS, protokol, ad.)
- PDU v rámci hierarchie zanořování PDU:



# Druhy chyb

- *K čemu může na úrovni paketů dojít?*
  - Ztráta (Zpoždění)
  - Ztráta fragmentovaných dat
  - Duplikace
  - Vložení (Zpoždění)
  - Změna pořadí

# Ztráta paketu

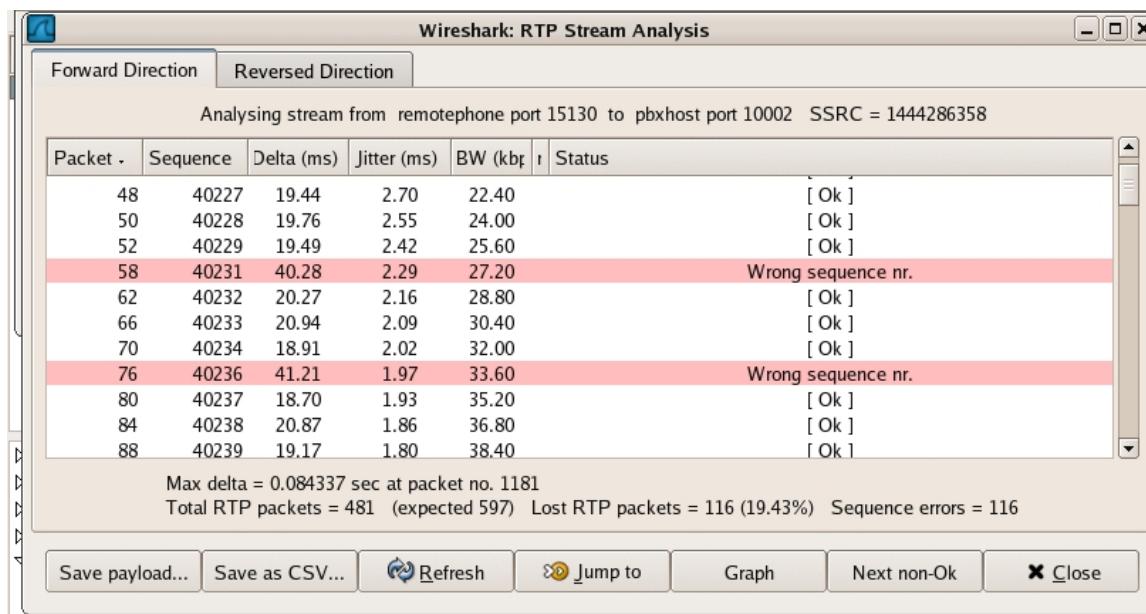
- Jedna z nejobvyklejších chyb v počítačových sítích
- Příčiny:
  - nejčastěji neopravitelné bitové chyby (jediný bit špatně stačí k tomu, aby byl celý paket zahozen)
  - zahlcení linky
  - Špatné směrovací tabulky
  - vadný HW nebo špatné ovladače síťového rozhraní

# Ztráta paketu – Illustrace

[https://ask.wireshark.org/uploads/ICCS\\_bad\\_SYN.png](https://ask.wireshark.org/uploads/ICCS_bad_SYN.png)

```
Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0
Ethernet II, Src: Netgear_b5:85:dc (c4:04:15:b5:85:dc), Dst: Apple_ae:le:86 (10:40:f3:ae:le:86)
  Destination: Apple_ae:le:86 (10:40:f3:ae:le:86)
  Source: Netgear_b5:85:dc (c4:04:15:b5:85:dc)
  Type: IP (0x0800)
  Padding: 0000
Frame check sequence: 0xcae2ff21 [incorrect, should be 0x885d6559]
  [FCS Good: False]
  [FCS Bad: True]
    [Expert Info (Error/Checksum): Bad checksum]
      [Bad checksum]
      [Severity level: Error]
      [Group: Checksum]
Internet Protocol Version 4, Src: 205.155.225.145 (205.155.225.145), Dst: 192.168.0.2 (192.168.0.2)
Transmission Control Protocol, Src Port: 443 (443), Dst Port: 62366 (62366), Seq: 0, Ack: 1, Len: 0
0000  10 40 f3 ae 1e 86 c4 04 15 b5 85 dc 08 00 45 20  .@..... E
0010  00 2c 90 0e 40 00 30 06 4a c6 cd 9b e1 91 c0 a8  ...@.0. J.....
0020  00 02 01 bb f3 9e 92 e1 3e 01 6d 61 bc 69 60 12  .....>.ma.i'.
0030  ff ff 38 87 00 00 02 04 05 64 00 00 ca e2 ff 21  ..8....d....
```

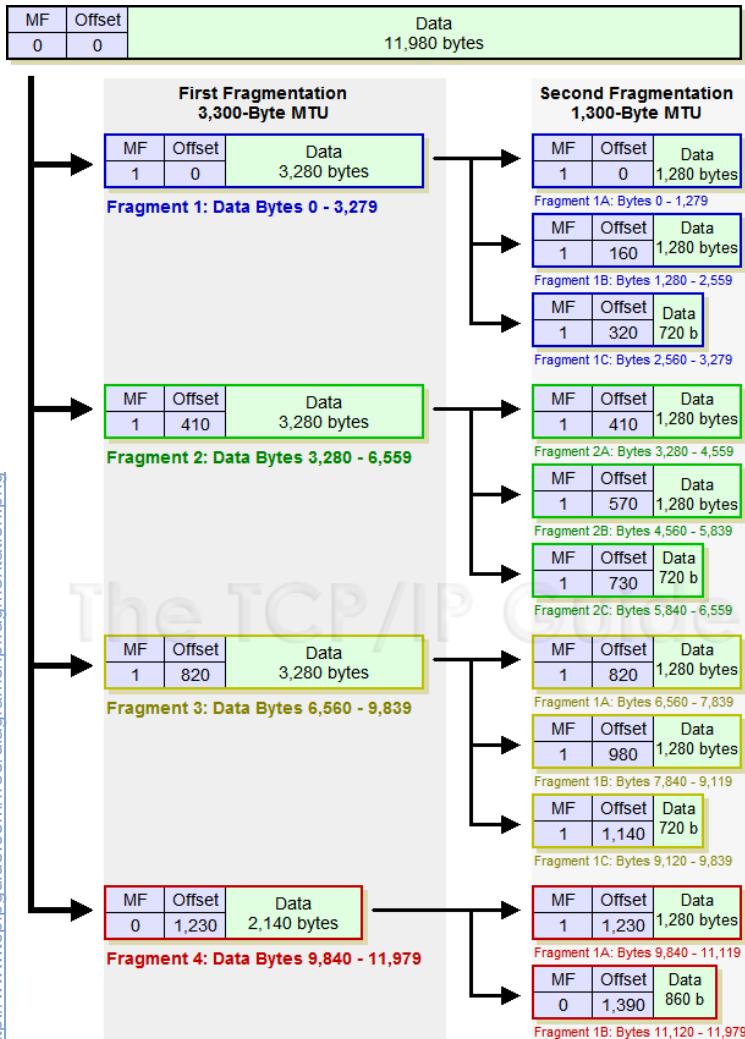
<http://www.linuxjournal.com/files/linuxjournal.com/lj/119398/19398f6.jpg>



# Ztráta fragmentovaných dat

- Limita MTU ovlivňuje i všechna do něj zanořená PDU
- Fragmentace
  - obvyklá u IPv4
  - u IPv6 „řeší“ PMTUD...tedy pokud zrovna funguje
- Příčina:
  - Jak dojde ke ztrátě fragmentu, dojde i ke ztrátě celého paketu!

# Fragmentace - Ilustrace



<http://www.tcpipguide.com/free/diagrams/lpf/fragmentation.png>

+ Frame 6 (1434 bytes on wire, 1434 bytes captured)  
 + Ethernet II, Src: Dell\_22:b9:3a (00:1c:23:22:b9:3a), Dst:   
 - Internet Protocol, src: 172.16.14.1 (172.16.14.1), Dst:   
 Version: 4  
 Header length: 20 bytes  
 + Differentiated Services Field: 0x00 (DSCP 0x00: Default)  
 Total Length: 1420  
 Identification: 0x1b21 (6945)  
 - Flags: 0x04 (Don't Fragment)  
 0... = Reserved bit: Not set  
 .1... = Don't fragment: Set  
 ..0... = More fragments: Not set  
 Fragment offset: 0  
 Time to live: 128  
 Protocol: TCP (0x06)  
 + Header checksum: 0x48b8 [correct]  
 Source: 172.16.14.1 (172.16.14.1)

[http://trycatch.be/cfs-filesystemfile.ashx/\\_key/CommunityServer.Blogs.Components.WeblogFiles/decaluwet/image\\_5F00\\_74B2E57C.png](http://trycatch.be/cfs-filesystemfile.ashx/_key/CommunityServer.Blogs.Components.WeblogFiles/decaluwet/image_5F00_74B2E57C.png)

# Duplikace paketu

- Příčina:
  - Ztratí se odeslaný paket nebo jeho potvrzení.
  - Odesilatel si myslí, že došlo k chybě při přenosu a znovuzasílá paket!

TCP	1514 [TCP segment of a reassembled PDU]
TCP	1514 [TCP Previous segment not captured] [TCP segment of a reassembled PDU]
TCP	1514 [TCP segment of a reassembled PDU]
TCP	54 49255→80 [ACK] Seq=205 Ack=5705681 Win=524288 Len=0
TCP	54 [TCP Dup ACK 5063#1] 49255→80 [ACK] Seq=205 Ack=5705681 Win=524288 Len=0
TCP	1514 [TCP segment of a reassembled PDU]
TCP	1514 [TCP segment of a reassembled PDU]
TCP	54 [TCP Dup ACK 5063#2] 49255→80 [ACK] Seq=205 Ack=5705681 Win=524288 Len=0
TCP	54 [TCP Dup ACK 5063#3] 49255→80 [ACK] Seq=205 Ack=5705681 Win=524288 Len=0
TCP	1514 [TCP segment of a reassembled PDU]
TCP	1514 [TCP segment of a reassembled PDU]
TCP	54 [TCP Dup ACK 5063#350] 49255→80 [ACK] Seq=205 Ack=5705681 Win=524288 Len=0
TCP	54 [TCP Dup ACK 5063#351] 49255→80 [ACK] Seq=205 Ack=5705681 Win=524288 Len=0
TCP	1514 [TCP segment of a reassembled PDU]
TCP	1514 [TCP segment of a reassembled PDU]
TCP	54 [TCP Dup ACK 5063#352] 49255→80 [ACK] Seq=205 Ack=5705681 Win=524288 Len=0
TCP	54 [TCP Dup ACK 5063#353] 49255→80 [ACK] Seq=205 Ack=5705681 Win=524288 Len=0
TCP	1514 [TCP Fast Retransmission] [TCP segment of a reassembled PDU]
TCP	54 49255→80 [ACK] Seq=205 Ack=6223981 Win=524288 Len=0
TCP	1514 [TCP segment of a reassembled PDU]

# Vložení paketu

- Příčina:

- Při přijetí zpožděného paketu, který dorazil k odesilateli po skončení jednoho a začátku dalšího separátního datového toku!
- Při podvrhávání paketu útočníkem snažícím se narušit integritu sítě.
- Pokud je paket chybami obzvláště výhodně zmrzačen, tak může být směrován špatnému příjemci.

# Vložení paketu - Ilustrace

Protocol	Length	Info
TDS	82	[TCP Previous segment not captured] Unknown Packet Type: 104 (Not last buffer)[Unreassembled Packet [incorrect TCP checksum]]
TCP	60	[TCP ACKed unseen segment] 36114 > 52157 [ACK] Seq=1333 Ack=236529 Win=258 Len=0[Malformed Packet]
TCP	82	[TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236557 Ack=1333 Win=4078 [TCP CHECKSUM INCORRECT] Len=0
TDS	82	[TCP Previous segment not captured] Unknown Packet Type: 86[Unreassembled Packet [incorrect TCP checksum]]
TCP	82	[TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236585 Ack=1333 Win=4078 [TCP CHECKSUM INCORRECT] Len=0
TDS	82	[TCP Previous segment not captured] Unknown Packet Type: 119 (Not last buffer)[Unreassembled Packet [incorrect TCP checksum]]
TCP	60	[TCP ACKed unseen segment] 45680 > 50624 [ACK] Seq=1347 Ack=214274 Win=65031 Len=0[Malformed Packet]
TCP	60	[TCP ACKed unseen segment] 36114 > 52157 [ACK] Seq=1333 Ack=236585 Win=258 Len=0[Malformed Packet]
TCP	82	[TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236613 Ack=1333 Win=4078 [TCP CHECKSUM INCORRECT] Len=0
TDS	82	[TCP Previous segment not captured] Unknown Packet Type: 39[Unreassembled Packet [incorrect TCP checksum]]
TCP	60	45680 > 50624 [ACK] Seq=1347 Ack=214330 Win=64975 Len=0[Malformed Packet]
TCP	69	[TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236641 Ack=1333 Win=4078 [TCP CHECKSUM INCORRECT] Len=0
TDS	69	[TCP Previous segment not captured] Unknown Packet Type: 81 (Not last buffer)[Unreassembled Packet [incorrect TCP checksum]]
TCP	60	[TCP ACKed unseen segment] 36114 > 52157 [ACK] Seq=1333 Ack=236641 Win=258 Len=0[Malformed Packet]
TCP	82	[TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236656 Ack=1333 Win=4078 [TCP CHECKSUM INCORRECT] Len=0
TDS	82	[TCP Previous segment not captured] Unknown Packet Type: 101 (Not last buffer)[Unreassembled Packet [incorrect TCP checksum]]
TCP	82	[TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236684 Ack=1333 Win=4078 [TCP CHECKSUM INCORRECT] Len=0
TDS	82	[TCP Previous segment not captured] Unknown Packet Type: 113[Unreassembled Packet [incorrect TCP checksum]]
TCP	60	[TCP ACKed unseen segment] 36114 > 52157 [PSH, ACK] Seq=1333 Ack=236656 Win=258 [TCP CHECKSUM INCORRECT] Len=2[Malformed Packet]
TCP	60	[TCP ACKed unseen segment] 45680 > 50624 [ACK] Seq=1347 Ack=214386 Win=64919 Len=0[Malformed Packet]
TCP	82	[TCP ACKed unseen segment] [TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236712 Ack=1337 Win=4077 [TCP CHECKSUM INCORRECT]
TCP	60	45680 > 50624 [PSH, ACK] Seq=1347 Ack=214401 Win=64904 [TCP CHECKSUM INCORRECT] Len=2[Unreassembled Packet [incorrect TCP checksum]]
TDS	82	[TCP ACKed unseen segment] [TCP Previous segment not captured] Unknown Packet Type: 81[Unreassembled Packet [incorrect TCP checksum]]
TCP	82	[TCP ACKed unseen segment] [TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236740 Ack=1337 Win=4077 [TCP CHECKSUM INCORRECT]
TCP	60	[TCP ACKed unseen segment] [TCP Previous segment not captured] 36114 > 52157 [ACK] Seq=1337 Ack=236712 Win=257 Len=0[Malformed Packet]
TCP	82	[TCP ACKed unseen segment] [TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236768 Ack=1337 Win=4077 [TCP CHECKSUM INCORRECT]
TCP	60	[TCP ACKed unseen segment] [TCP Previous segment not captured] 45680 > 50624 [ACK] Seq=1351 Ack=214457 Win=64848 Len=0[Malformed Packet]
TDS	82	[TCP ACKed unseen segment] [TCP Previous segment not captured] Unknown Packet Type: 23 (Not last buffer)[Unreassembled Packet [incorrect TCP checksum]]
TCP	60	[TCP ACKed unseen segment] 36114 > 52157 [ACK] Seq=1337 Ack=236768 Win=259 Len=0[Malformed Packet]
TCP	82	[TCP Previous segment not captured] 52157 > 36114 [PSH, ACK] Seq=236796 Ack=1337 Win=4077 [TCP CHECKSUM INCORRECT] Len=0

[https://ask.wireshark.org/upfiles/WireShark\\_bad\\_packets\\_1.png](https://ask.wireshark.org/upfiles/WireShark_bad_packets_1.png)

# Změna pořadí

- Příčina:
  - Příjemce přijal pakety v jiném pořadí, než je odesilatel odeslal!
- Problémem packed-switch sítí
  - Dynamické směrování – obvykle není garantována cesta
  - Pakety od odesilatele mohou být směrovány různými cestami, a tak dorazí v rozdílných časech

No.	Time	Source	Destination	Protocol	Length	Info
562	4.618781	192.168.129.167	72.5.124.61	TCP	54	1230-80 [ACK] Seq=4023 Ack=216433 Win=64240 Len=0
563	4.619246	72.5.124.61	192.168.129.167	TCP	646	[TCP Previous segment not captured] [TCP segment of a reassembled PDU]
564	4.619329	192.168.129.167	72.5.124.61	TCP	66	[TCP Dup ACK 562#1] 1230-80 [ACK] Seq=4023 Ack=216433 Win=64240 Len=0
565	4.620247	72.5.124.61	192.168.129.167	TCP	1514	[TCP out-of-order] [TCP segment of a reassembled PDU]
566	4.620410	72.5.124.61	192.168.129.167	TCP	1514	[TCP segment of a reassembled PDU]
567	4.620809	72.5.124.61	192.168.129.167	TCP	1514	[TCP segment of a reassembled PDU]
568	4.620880	192.168.129.167	72.5.124.61	TCP	54	1231-80 [ACK] Seq=3780 Ack=129763 Win=64240 Len=0
569	4.621768	72.5.124.61	192.168.129.167	TCP	1514	[TCP segment of a reassembled PDU]
570	4.622292	72.5.124.61	192.168.129.167	TCP	1514	[TCP out-of-order] [TCP segment of a reassembled PDU]
571	4.622378	192.168.129.167	72.5.124.61	TCP	54	1230-80 [ACK] Seq=4023 Ack=219945 Win=64240 Len=0

<https://blog.packet-foo.com/wp-content/uploads/2014/08/TCPExpertWireshark1.12.png>

# Měřitelnost

- PER...packet error rate [%]
- BER...bit error rate [%]
- $N$ ...délka paketů v [bitech]
- $p_p$ ...odhadovaný PER v [%]
- $p_e$ ...pravděpodobnost vzniku bitové chyby, tj. odhadovaný BER [%]

$$PER = \frac{[\text{počet chybných paketů}]}{[\text{celkový počet přenesených paketů}]}$$

$$BER = \frac{[\text{počet chybných bitů}]}{[\text{celkový počet přenesených bitů}]}$$

$$p_p = 1 - (1 - p_e)^N$$

- Příklad:

- Spočítejte  $p_p$  pro přenos 1400 B dlouhého paketu při  $p_{e1} = 1\%$  a  $p_{e2} = 0,01\%$
- $p_{p1} = 1 - (1 - 0.01)^{1400 \cdot 8} \doteq \underline{\underline{99,999 \%}}$
- $p_{p2} = 1 - (1 - 0.0001)^{1400 \cdot 8} \doteq \underline{\underline{67,374 \%}}$

# Detekce chyb

- *Jak tedy detektovat paketové chyby?*

# Sekvenční čísla

- $\stackrel{\text{def}}{=}$  jedinečné číslo v paketu, které identifikuje jeho pořadí v rámci datového toku
- Chování:
  - inkrementováno pro každý další (avšak ne pro znova odesílaný) paket
  - při přetečení se „nuluje“ (sek.čísla tvoří okruh)
- Co to přináší za řešení?
  - v datovém toku se pozná duplicita a změna pořadí
  - ztráta paketu je vnímána jako mezera v posloupnosti
  - vložení paketu se může (ale bohužel i nemusí) projeví jako odchylka od posloupnosti

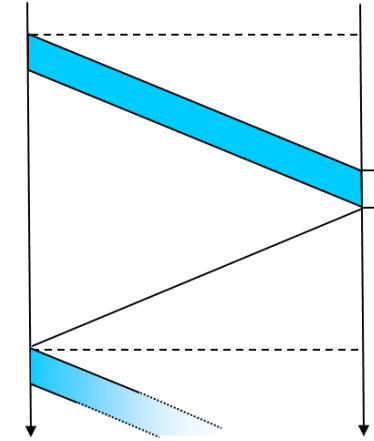
+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

-	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

# Velikost sekvenčního čísla ①

- Prostor sekvenčních čísel je omezený
  - $n$  bitový typ jen  $2^n$  čísel
  - např. pro  $n = 3$  jen 8 sekvenčních čísel
  - pro TCP je  $n = 32$
- Jak k tomu ale došli aneb parametry pro odvození prostoru sekvenčních čísel?
  - $MPL$ ...maximální povolený čas existence paketu v síti v [s]
  - $T$ ...maximální doba, po kterou je odesilatel vyčkávající na potvrzení schopen mít paket připravený k znovuzaslání v [s]
  - $A$ ...maximální doba, po kterou se příjemce může zdržet před zasláním potvrzení v [s]
  - $R$ ... maximální přenosová rychlosť odesilatel v  $\left[ \frac{\text{pakety}}{\text{s}} \right]$



# Velikost sekvenčního čísla (2)

- Jaké je největší možné množství paketů, které může být vygenerováno zatímco je jeden paket aktivní (buď čekající na znovuzaslání, či na potvrzení)?

$$(2MPL + T + A) \cdot R$$

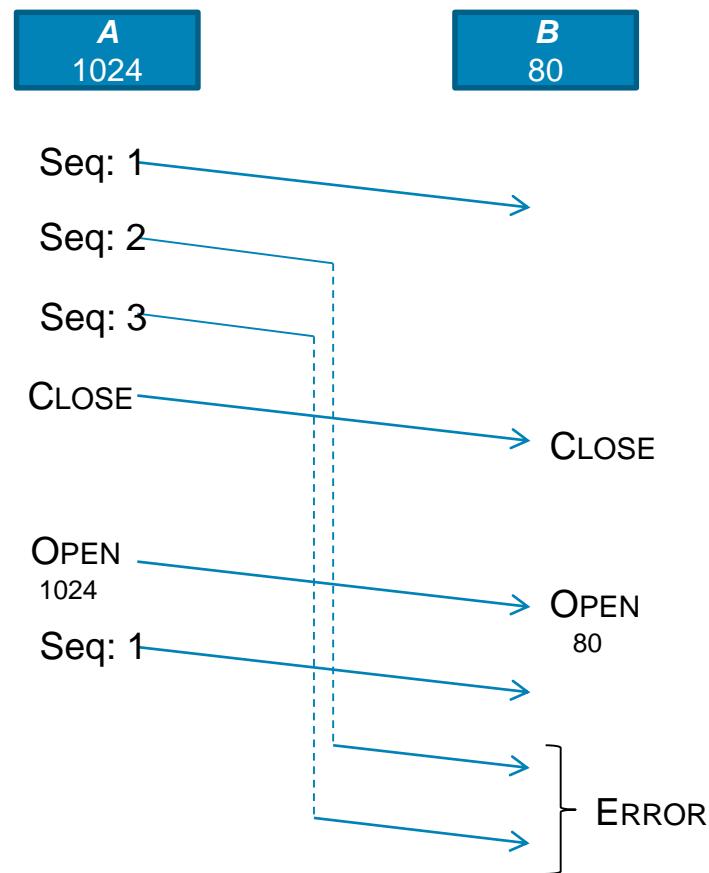
- Příklad:
  - Paket může „žít“ v síti maximálně 2 minuty. Odesilatel má linku o propustnosti 2 Mbps a je ochoten paket znovuzasílat po dobu 1 minuty. Příjemce vygeneruje potvrzení do 500 ms od přijetí. Jaký je nejmenší prostor sekvenčních čísel, jestliže pakety mají velikost 40 bytů?
  - $MPL = 120; T = 60; A = 0,5; R = \frac{2 \cdot 10^6}{40.8} = 6\,250$
  - $(2MPL + T + A) \cdot R = (240 + 60 + 0,5) \cdot 6250 = 1\,878\,125$
  - $n: 2^n > (2MPL + T + A) \cdot R$ , tzn.  $n = \log_2 1878125 \doteq \underline{\underline{21}}$

# Vložení paketu a MPL ①

## Scénář

- V rámci jednoho datového toku si počítač **A** (z portu 1024) povídá s počítačem **B** (na portu 80) a začnou číslovat se sekvenčním číslem 1. Dojde ke zpoždění paketů na cestě. V rámci dalšího datového toku **A** i **B** použijí ~~stejná~~ portová čísla a stejná sekvenční čísla. Do jejich konverzace se připletou pakety z předchozího toku →

## Illustrace



# Vložení paketu a MPL ②

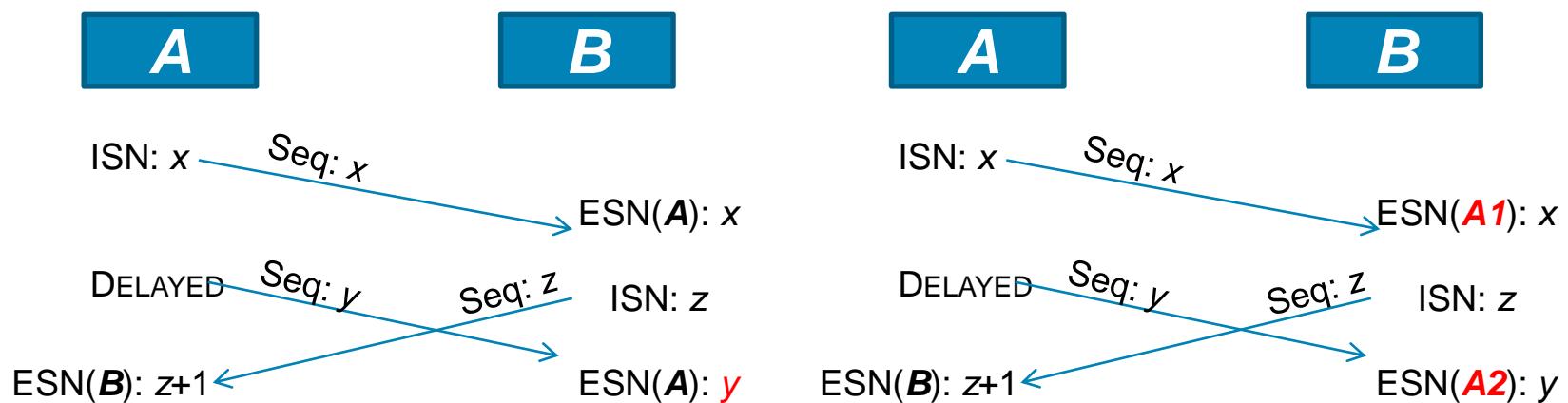
- Obvyklá hodnota MPL je 15 s až 2 min
- Řešení:
  - **inkarnační číslo** (označuje jedinečně datový tok) – ☹
  - znovupoužití portu až po uplynutí jedné MPL – ☺
  - sériové přiřazování dynamických portů + generování sekvenčního čísla z registru, který je inkrementován každý tik hodin – ☺

# Handshake

- Zahájení komunikace:
  - Na jedné straně vygenerování ISN na druhé straně jeho uložení do ESN a následné používání pro spojení.
  - Používá se tzv. **synchronizačních SYN paketů**
- Ukončení komunikace:
  - Dojde k uvolnění ESN a ISN pro daný datový tok
  - Terminující strana použije **finalizačního FIN paketu**

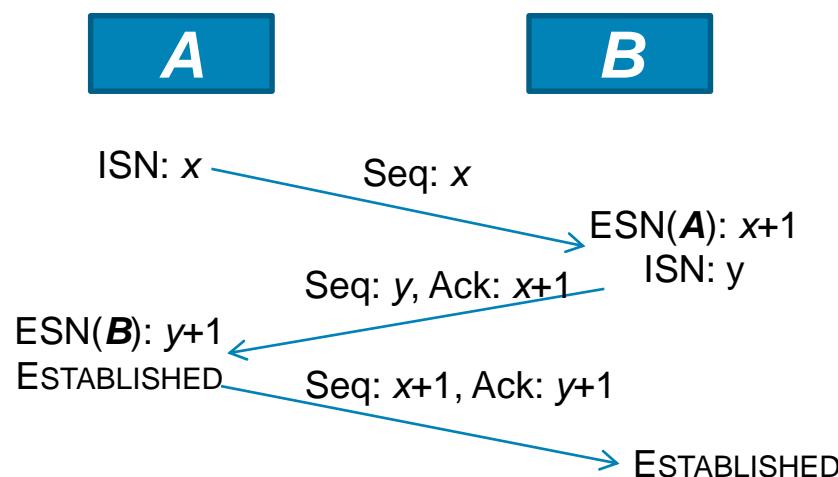
# 2Way Handshake

- Princip:
  - Pošli své sekvenční číslo!
- Nevýhoda:
  - Neumí se vyrovnat se zpožděnými SYN pakety (přepis ESN či nové spojení)
  - Zpožděná data můžou vést k problému vložení

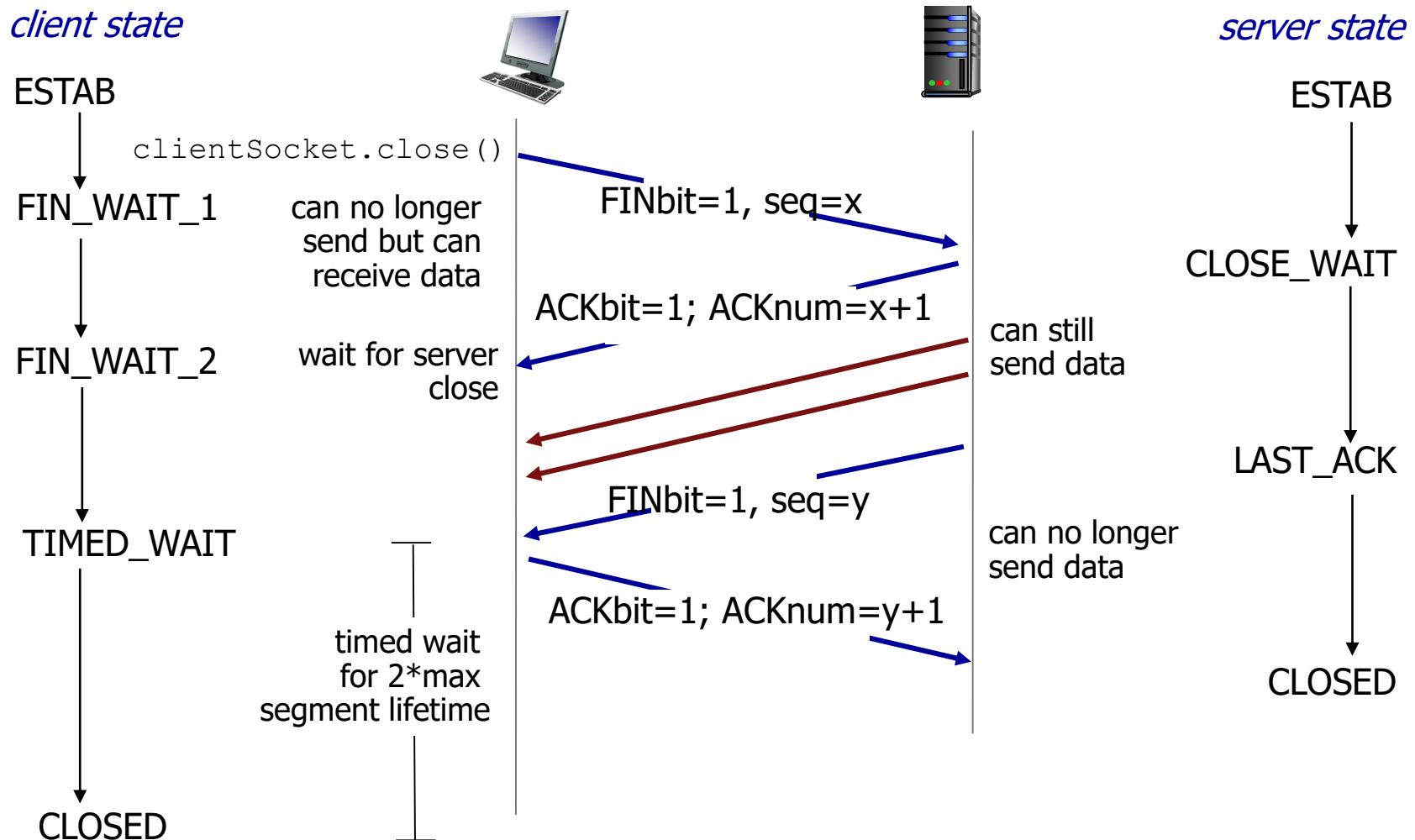


# 3Way Handshake

- Princip:
  - Pošli své sekvenční číslo a potvrď cizí!
- Princip používaný v TCP – ten potvrzuje číslem následujícího paketu, obecně lze však potvrzovat i dorazivší paket



# Akceptace po skončení spojení?



# Detekce ztráty paketu

- Dvě techniky:
  - **Timeout**
  - **Negativní potvrzování**

# Detekce ztráty paketu – Timeout ①

- Timeouty na různých stranách
  - odesílatel vs. příjemce
- *Problémem je jak zvolit jeho hodnotu!*
- Může být stále fixní – ☹
- Může být odvozený od **round-trip time (RTT)**, a to
  - fixně z poslední naměřené hodnoty RTT – ☺
  - skrz zjemňující průměr měření RTT – ☺
  - [RFC 2988](#) a navazující [RFC 6298](#)

# Detekce ztráty paketu – Timeout (2)

## ■ Exponential Weighted Moving Average

- $rtt_k$  ... hodnota RTT pro  $k$ -tý paket
- $a$  ... parametr v intervalu  $\langle 0; 1 \rangle$
- $srtt_k$  ... zjedněný RTT odhad pro  $k$ -tý paket
- $timeout_k$  ... vypočtená hodnota timeoutu pro  $k$ -tý paket

$$srtt_k = (1 - a) \cdot srtt_{k-1} + a \cdot rtt_{k-1}$$
$$timeout_k = srtt_k$$

- Výpočet nejvíce ovlivňuje volba  $a$  a  $srtt_0$ !!!
  - doporučená  $a = 0,125$

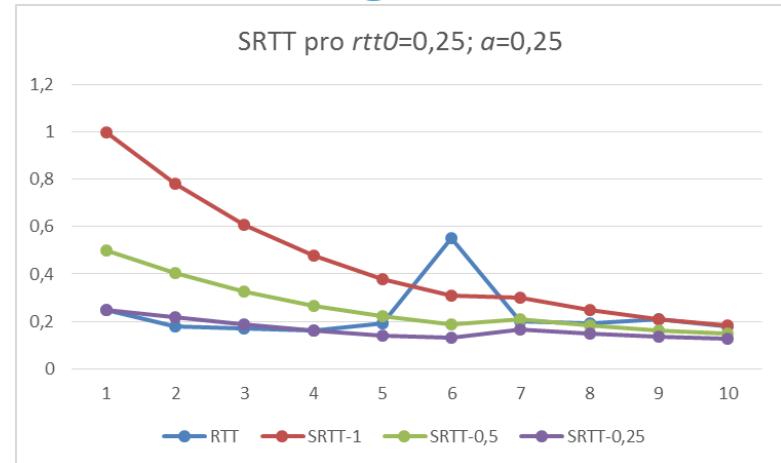
# Detekce ztráty paketu – Timeout ③

- Výsledky pro rozdílná

$rtt_0 = \{1\text{s}, \frac{1}{2}\text{s}, \frac{1}{4}\text{s}\}, a = 0,25,$

s různými  $srtt_0 = \{1\text{s}, \frac{1}{2}\text{s}, \frac{1}{4}\text{s}\}$

při RTT linky 0,185



	$rtt_0 = 1$	$srtt_0 = 1$	$srtt_0 = \frac{1}{2}$	$srtt_0 = \frac{1}{4}$	$rtt_0 = \frac{1}{2}$	$srtt_0 = 1$	$srtt_0 = \frac{1}{2}$	$srtt_0 = \frac{1}{4}$	$rtt_0 = \frac{1}{4}$	$srtt_0 = 1$	$srtt_0 = \frac{1}{2}$	$srtt_0 = \frac{1}{4}$
	$rtt_k$	$srtt_k$	$srtt_k$	$srtt_k$	$rtt_k$	$srtt_k$	$srtt_k$	$srtt_k$	$rtt_k$	$srtt_k$	$srtt_k$	$srtt_k$
$t_1$	0,180	0,875	0,500	0,313	0,180	0,813	0,438	0,250	0,180	0,781	0,406	0,219
$t_2$	0,170	0,679	0,398	0,257	0,170	0,632	0,351	0,210	0,170	0,608	0,327	0,187
$t_3$	0,160	0,530	0,319	0,214	0,160	0,495	0,284	0,179	0,160	0,478	0,267	0,161
$t_4$	0,190	0,418	0,260	0,180	0,190	0,391	0,233	0,154	0,190	0,378	0,220	0,141
$t_5$	0,550	0,337	0,218	0,159	0,550	0,317	0,199	0,139	0,550	0,307	0,189	0,129
$t_6$	0,200	0,322	0,233	0,188	0,200	0,307	0,218	0,173	0,200	0,299	0,210	0,166
$t_7$	0,190	0,266	0,199	0,166	0,190	0,255	0,188	0,155	0,190	0,249	0,183	0,149
$t_8$	0,210	0,223	0,173	0,148	0,210	0,215	0,165	0,140	0,210	0,211	0,161	0,136
$t_9$	0,180	0,194	0,156	0,137	0,180	0,188	0,150	0,131	0,180	0,184	0,147	0,128

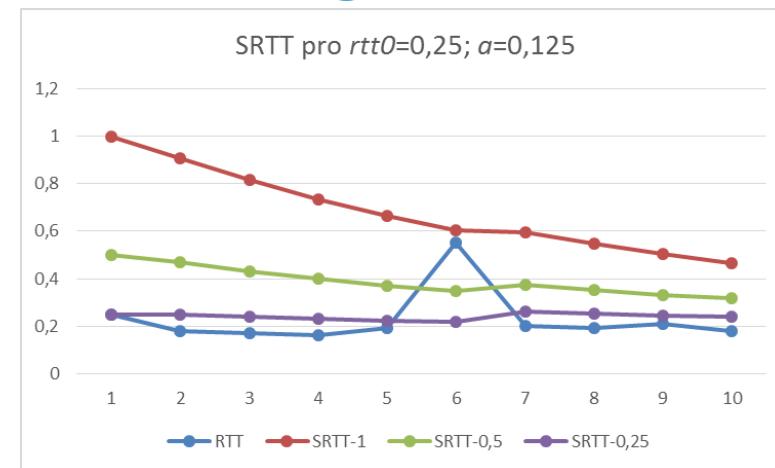
# Detekce ztráty paketu – Timeout ④

- Výsledky pro rozdílná

$rtt_0 = \{1\text{s}, \frac{1}{2}\text{s}, \frac{1}{4}\text{s}\}$ ,  $a = 0,125$ ,

s různými  $srtt_0 = \{1\text{s}, \frac{1}{2}\text{s}, \frac{1}{4}\text{s}\}$

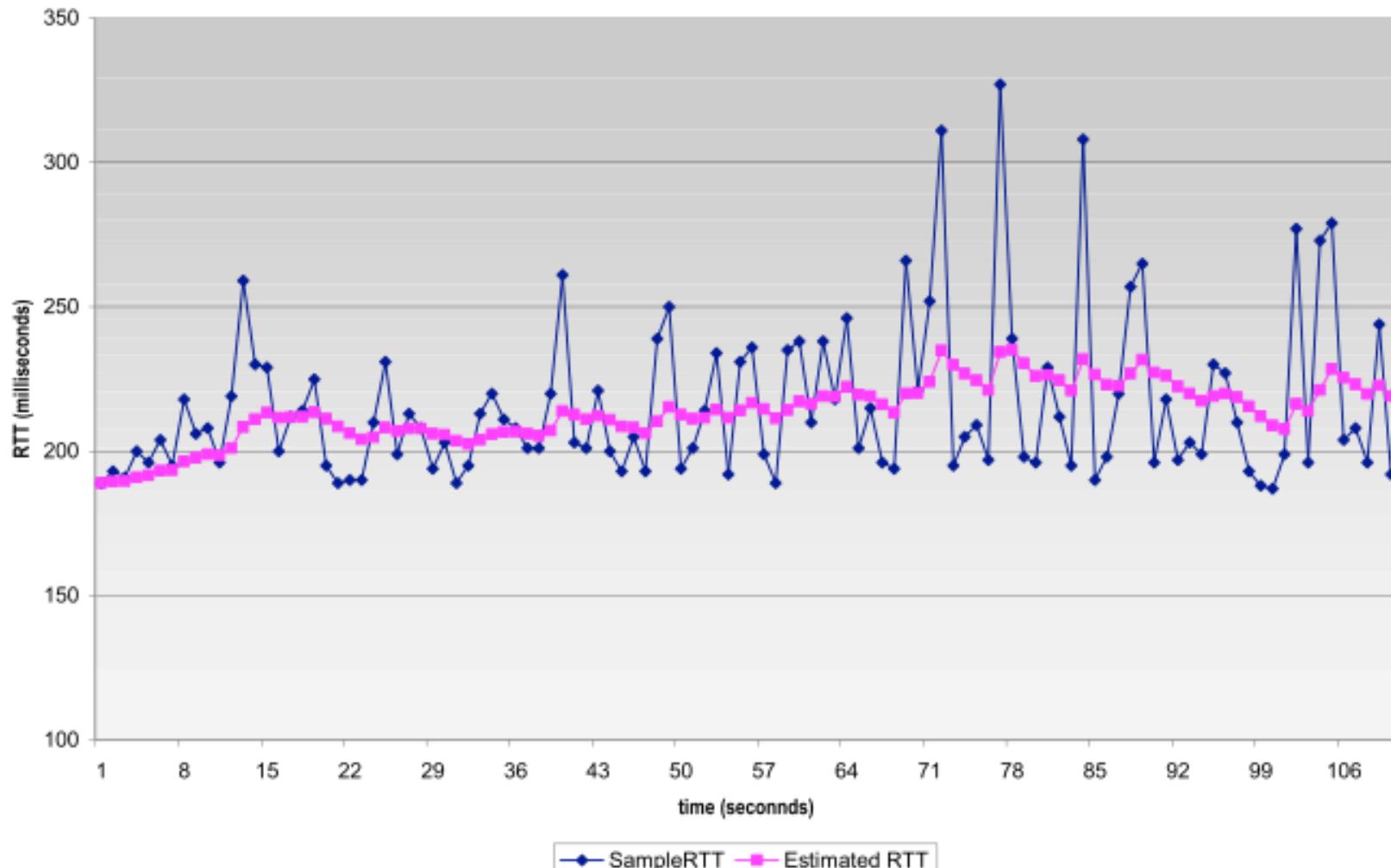
při RTT linky 0,185



	$rtt_0 = 1$	$srtt_0 = 1$	$srtt_0 = \frac{1}{2}$	$srtt_0 = \frac{1}{4}$	$rtt_0 = \frac{1}{2}$	$srtt_0 = 1$	$srtt_0 = \frac{1}{2}$	$srtt_0 = \frac{1}{4}$	$rtt_0 = \frac{1}{4}$	$srtt_0 = 1$	$srtt_0 = \frac{1}{2}$	$srtt_0 = \frac{1}{4}$	
	$rtt_k$	$srtt_k$	$srtt_k$	$srtt_k$	$rtt_k$	$srtt_k$	$srtt_k$	$srtt_k$	$rtt_k$	$srtt_k$	$srtt_k$	$srtt_k$	
	1	1	0,5	0,25	0,5	1	0,5	0,25	0,25	0,25	1	0,5	0,25
$t_1$	0,180	1,000	0,563	0,344	0,180	0,938	0,500	0,281	0,180	0,906	0,469	0,250	
$t_2$	0,170	0,898	0,515	0,323	0,170	0,843	0,460	0,269	0,170	0,815	0,433	0,241	
$t_3$	0,160	0,807	0,472	0,304	0,160	0,759	0,424	0,256	0,160	0,735	0,400	0,232	
$t_4$	0,190	0,726	0,433	0,286	0,190	0,684	0,391	0,244	0,190	0,663	0,370	0,223	
$t_5$	0,550	0,659	0,402	0,274	0,550	0,622	0,366	0,237	0,550	0,604	0,347	0,219	
$t_6$	0,200	0,645	0,421	0,309	0,200	0,613	0,389	0,277	0,200	0,597	0,373	0,260	
$t_7$	0,190	0,590	0,393	0,295	0,190	0,561	0,365	0,267	0,190	0,547	0,351	0,253	
$t_8$	0,210	0,540	0,368	0,282	0,210	0,515	0,343	0,257	0,210	0,503	0,331	0,245	
$t_9$	0,180	0,498	0,348	0,273	0,180	0,477	0,327	0,251	0,180	0,466	0,316	0,241	

# Ilustrace odhadu

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr



# Detekce ztráty paketu – Timeout ⑤

- Ještě lepšího odhadu se dosahuje **EWMA se zakomponovanou odchylkou rozptylu** (mean deviation):
  - $e_k$ ... hodnota rozptylu pro  $k$ -tý paket
  - $rttvar_k$ ... hodnota odchylky se zakomponovaným rozptylem pro  $k$ -tý paket
  - $b$ ... konstanta udávající variabilitu RTT v čase v intervalu  $\langle 0; 1 \rangle$
  - doporučené  $b = 0,25$

$$srtt_k = (1 - a) \cdot srtt_{k-1} + a \cdot rtt_{k-1}$$

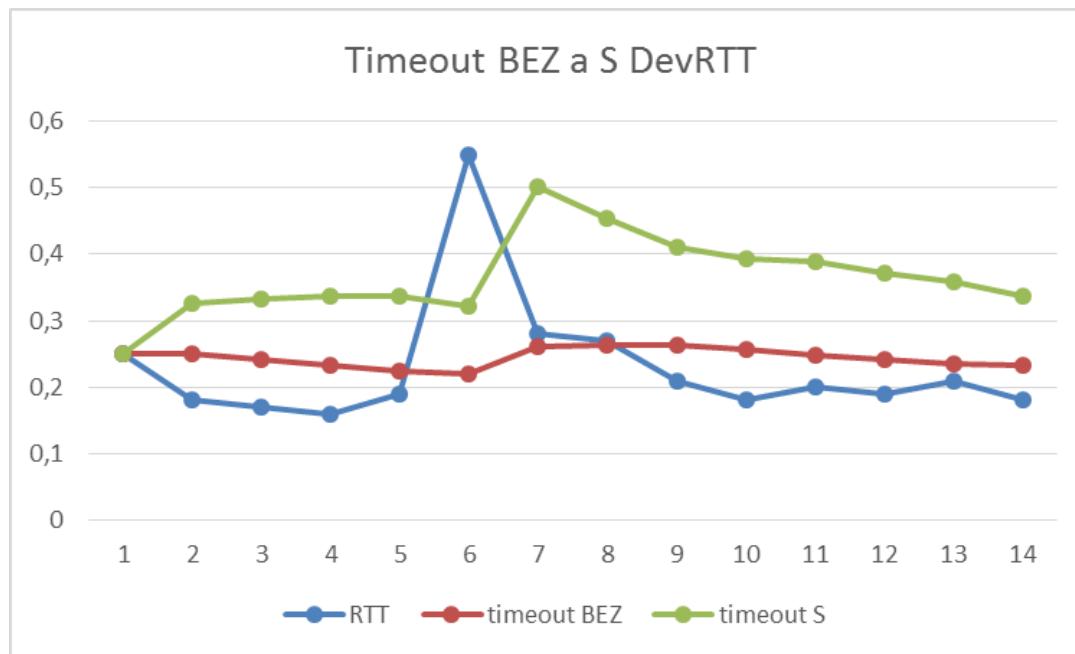
$$e_k = |srtt_{k-1} - rtt_{k-1}|$$

$$rttvar_k = (1 - b) \cdot rttvar_{k-1} + b \cdot e_{k-1}$$

$$timeout_k = srtt_k + \textcolor{red}{X} rttvar_k$$

# Detekce ztráty paketu – Timeout ⑥

- $a = 0,125 \ b = 0,25 \ X=4$
- [s]  $rtt_0 = 0,25$  a  $rttvar_0 = 0,05$
- ideální RTT 0,185



$rtt_0 = \frac{1}{4}$	$srtt_0 = \frac{1}{4}$	$e_k$	$rttvar_k$	$timeout_k$
$rtt_k$	$srtt_k$			
0,25	0,25	0	0,05	0,250
0,180	0,250	0,0700	0,0375	0,325
0,170	0,241	0,0713	0,0456	0,333
0,160	0,232	0,0723	0,0520	0,336
0,190	0,223	0,0333	0,0571	0,338
0,550	0,219	0,3309	0,0512	0,321
0,280	0,260	0,0195	0,1211	0,503
0,270	0,263	0,0071	0,0957	0,454
0,210	0,264	0,0538	0,0735	0,411
0,180	0,257	0,0771	0,0686	0,394
0,200	0,247	0,0475	0,0707	0,389
0,190	0,242	0,0515	0,0649	0,371
0,210	0,235	0,0251	0,0616	0,358
0,180	0,232	0,0519	0,0524	0,337

# Detekce ztráty paketu – Negativní potvrzování

- Funkční bez jakýchkoli timeoutů
- Princip:
  - Příjemce odesilateli zašle, které všechny pakety k němu nedorazily (v případě, že detekuje „mezeru“ v sekvenčních číslech)
- Nevýhody:
  - Uvažujme, že pakety bývají zahazovány v případě zahlcení sítě. Pak posílání NACKů jen dále přispívá k vyčerpávání už tak zahlcených síťových prostředků.
  - *A co když se ztratí v síti samotný NACK?*

# Korekce chyb

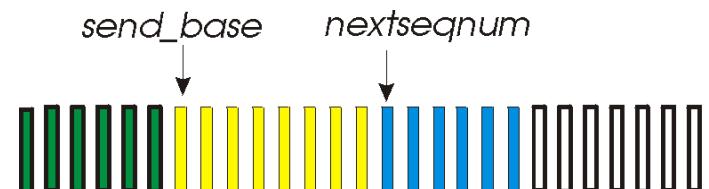
- Už víme, že pomocí sekvenčních čísel + timeoutů + 3way handshakeu celkem úspěšně zdetekujeme chybu paketu...
- ...jak ji však vyrovnat (korigovat)?

# Znovuzaslání

- a.k.a. **retransmission**
- *Tím, že chybějící pakety pošleme znovu!*
- **Automatic Repeat/Request (ARQ)**
  - Čeká se na ACK; když nepřijde do timeoutu, dojde k znovuzaslání
- Strategie, které zjišťují, co znova poslat
  - **Stop-and-Wait ARQ**
  - **Go-Back-N ARQ**
  - **Selektivní znova zaslání (selective repeat) ARQ**
  - **SMART**

# Chyby-kontrolující okno

- (aka **error-control window**)  $\stackrel{\text{def}}{=}$  nejmenší podmnožina z prostoru sekvenčních čísel odeslaných paketů, které ještě nebyly potvrzeny, kde počet prvků množiny je **velikost okna** (parametr  $w$ )



- Pokaždé když odesilatel:

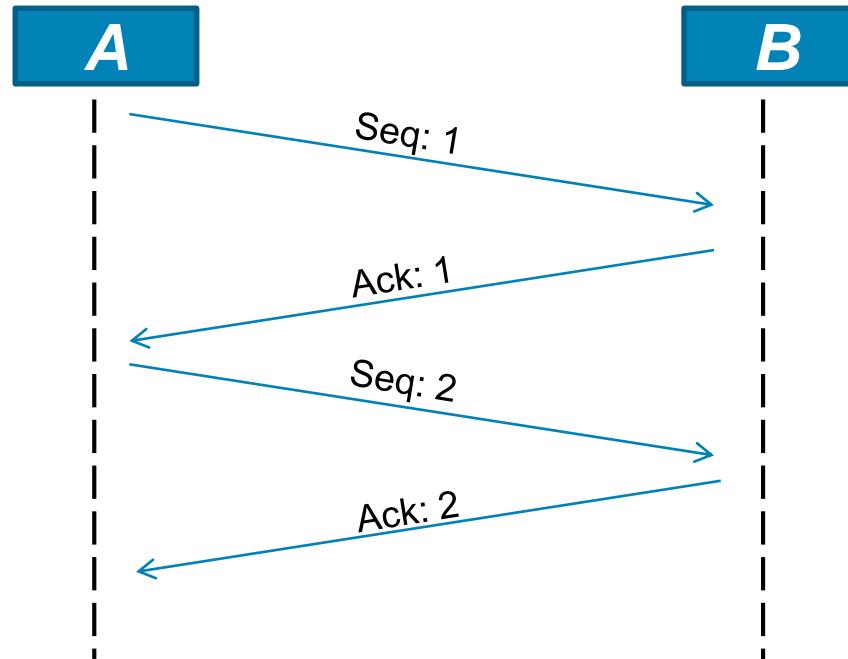
- příjme potvrzení → okno se „smrskne zleva“
- odešle paket → okno se „rozšíří zprava“



- Tomuto principu se říká **klouzavé okno** (aka sliding-window)

# Stop-and-wait

- Zjednodušený princip klouzavého okna o velikosti 1

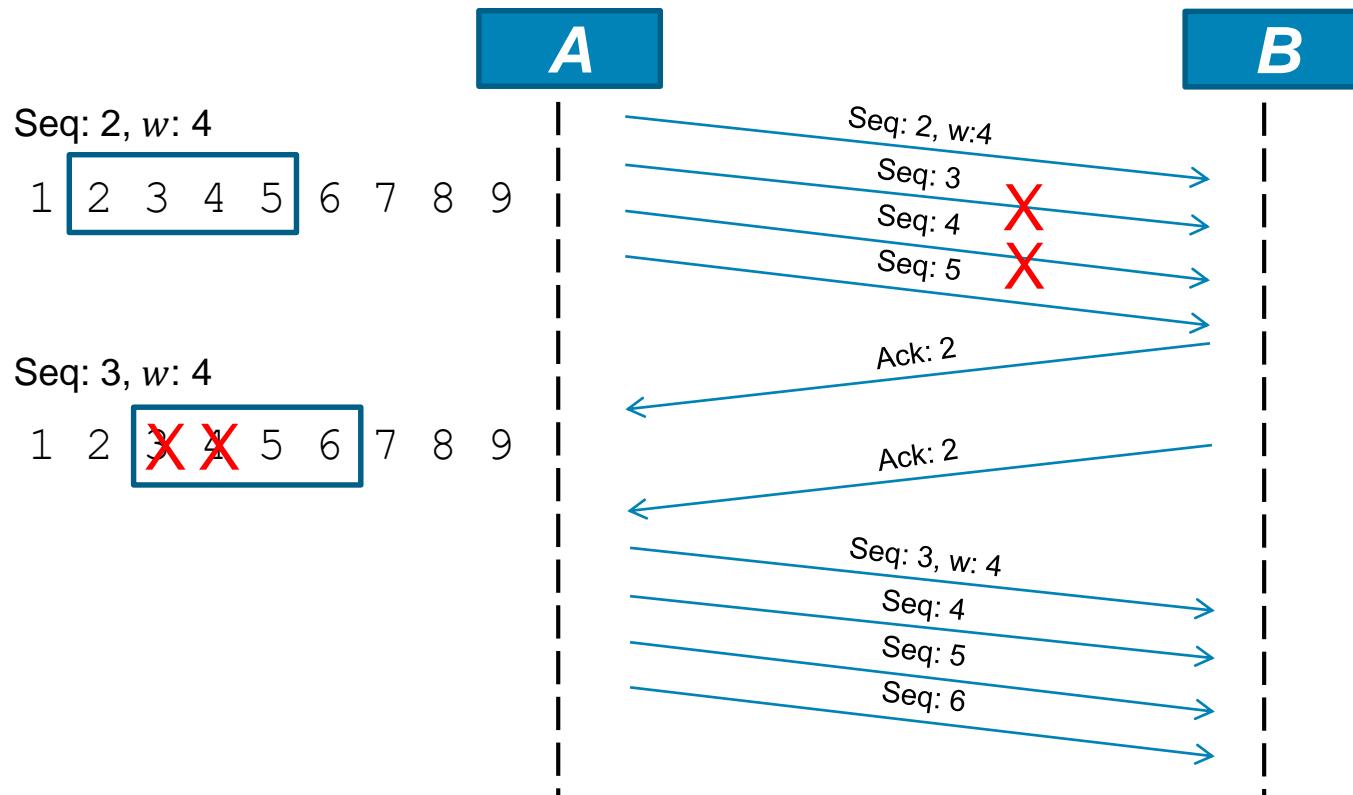


- Nevýhody:
  - Oproti ostatním špatná efektivita při využití pásma

# Go-Back-N ①

- Princip:

- Příjemce potvrzuje jen poslední korektně přijatý paket z okna.

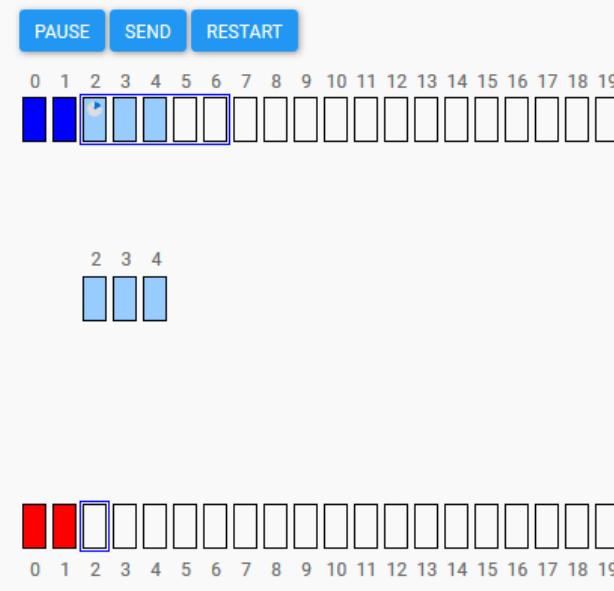


# Go-Back-N: Demo

- <http://nes.fit.vutbr.cz/ivesely/pds/>

Go-Back-N   Selective Repeat - Fast Retransmit   Selective Repeat - Bit mask   Selective Repeat - NACK   SMART

PAUSE   SEND   RESTART



speed 1.00 + -

Send window size  
5

Packet speed to receiver  
5000

Packet speed to sender  
5000

Packet timeout  
12000

Legend:  
Packet (blue square)  
Received (red square)  
Ack (yellow square)  
Ack received (blue square with outline)

- The receiver confirms only the last correctly received packets of window.
- Sender send all not acknowledged packet after timeout

# Go-Back-N ②

- Výhody:
  - jednoduchý (přijímá pakety v pořadí a odmítá je mimo něj, takže není potřeba žádného bufferu)
  - konzervativní (pokud se chyby objevují v burstech)
- Nevýhody:
  - plýtvá šířkou pásma (znovu se zasílají i dříve úspěšně poslané pakety), což může přispívat k zahlcování sítě
  - Při chybách neumožňuje nikdy využít celé šířky pásma!!!

# Go-Back-N ③

- $p$  ... pravděpodobnost ztráty paketu v [%]
- $w$  ... šířka okna v [paketech]
- $eff$  ... maximální možná efektivita linky, což je procento z šířky pásma, které se dá využít v případě, kdy nedochází k chybám

$$eff = \frac{1 - p}{1 - p + pw}$$

# Go-Back-N ④

- Příklad:
  - Spočítejte maximální efektivitu linky s Go-Back-N algoritmem znovuzaslání mezi uzly, kde šířka okna je 250 paketů při pravděpodobnosti chyby 1 % a 0,01 %!
  - $eff_1 = \frac{0,99}{0,99+2,5} \doteq \underline{\underline{28,367 \%}}$
  - $eff_2 = \frac{0,9999}{0,9999+0,025} \doteq \underline{\underline{97,561 \%}}$
  - Go-Back-N je tedy tím NEvhodnější, čím je bud' linka rychlejší nebo pravděpodobnost chyb větší!!!

# Selektivní znovu zasílání

- Existuje několik variant používající:
  - Fast Retransmit
  - Bitovou masku
  - NACK
- Obecné výhody:
  - chytřejší než Go-Back-N
  - neplýtvá šírkou pásma (zasílány jsou jen ty pakety, které se ztratily)
- Obecné nevýhody
  - složitější implementace (buffery na řazení paketů, paměť co už dorazilo a co ne)

# Selective Repeat – FastReXmit: Demo

- <http://nes.fit.vutbr.cz/ivesely/pds/>

Go-Back-N    Selective Repeat - Fast Retransmit    Selective Repeat - Bit mask    Selective Repeat - NACK    SMART

PAUSE    SEND    RESTART

speed 1.00 + -

Send window size  
4

Packet speed to receiver  
5000

Packet speed to sender  
5000

Sender timeout  
30000

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

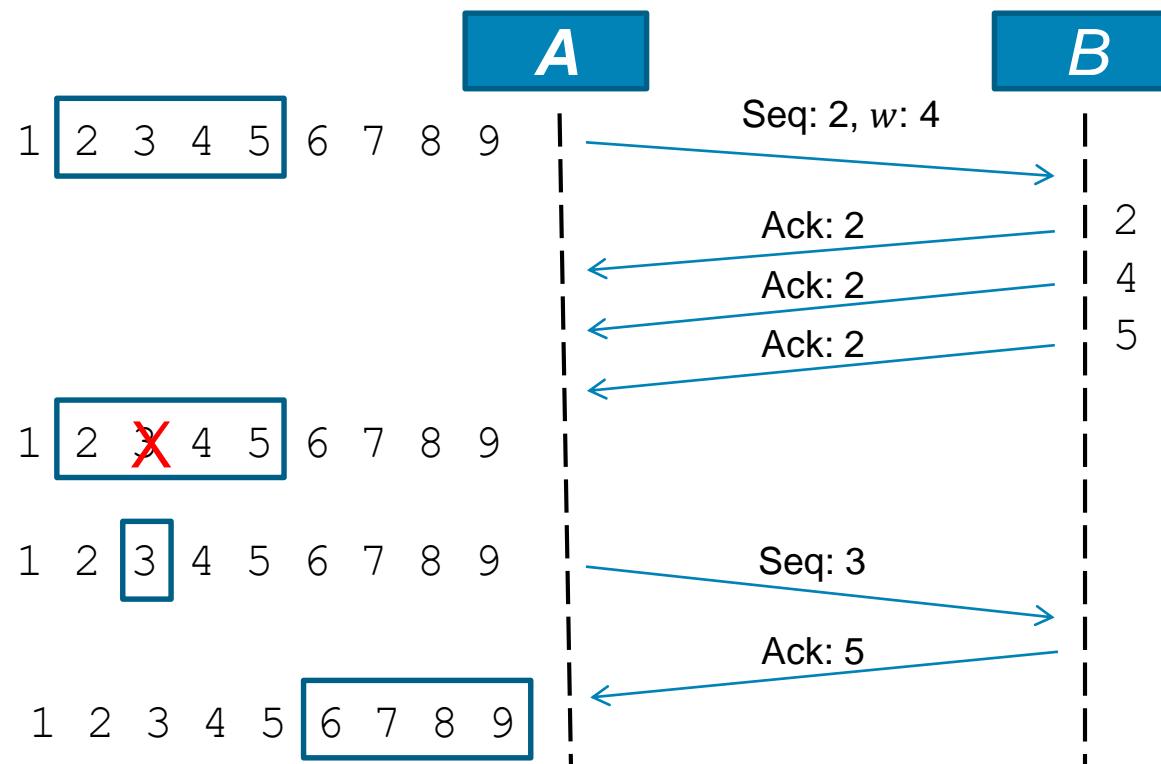
Packet  
Received  
Ack  
Ack received

- The receiver confirms only the last correctly received packets of windows.
- Sender send all not acknowledged packet after timeout.

# Selektivní znovu zasílání – Fast Retransmit ①

## ■ Princip:

- V rámci okna se potvrzuje každý přijatý paket v pořadí zvlášť, přičemž každý následující (**kumulativní**) ACK obsahuje sekvenční číslo posledně korektně přijatého paket.

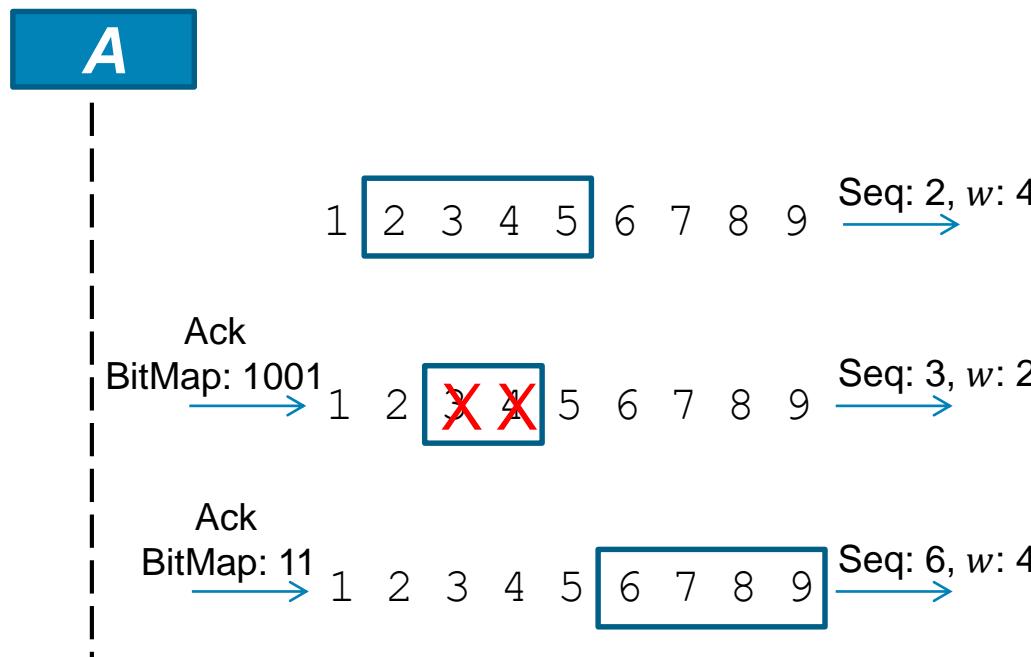


# Selektivní znovu zasílání – Fast Retransmit ②

- Výhoda:
  - varianta používaná v TCP
  - [http://media.pearsoncmg.com/aw/ecs\\_kurose\\_comnetwork\\_6/video\\_applets/SRindex.html](http://media.pearsoncmg.com/aw/ecs_kurose_comnetwork_6/video_applets/SRindex.html)
- Nevýhody:
  - vhodná jen pro relativně spolehlivé linky, kde nedochází k více než jedné paketové chybě v rámci chyby-kontrolujícího okna

# Selektivní znovu zasílání – Bitová maska

- Princip:
  - Pomocí bitové masky dává příjemce odesilateli vědět, které pakety z aktivního chyby-kontrolujícího okna dorazily v pořádku a které nikoli



- Nevýhody:
  - zvětšení hlavičky ACK musí v ní být místo pro bitovou mapu

# Selective Repeat – BitMask: Demo

- <http://nes.fit.vutbr.cz/ivesely/pds/>

Go-Back-N   Selective Repeat - Fast Retransmit   **Selective Repeat - Bit mask**   Selective Repeat - NACK   SMART

---

PAUSE   SEND   RESTART

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P																			

1001

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R																			

---

■ Packet  
■ Received  
■ Ack  
■ Ack received

speed 1.00 + -

Send window size  
4

Packet speed to receiver  
5000

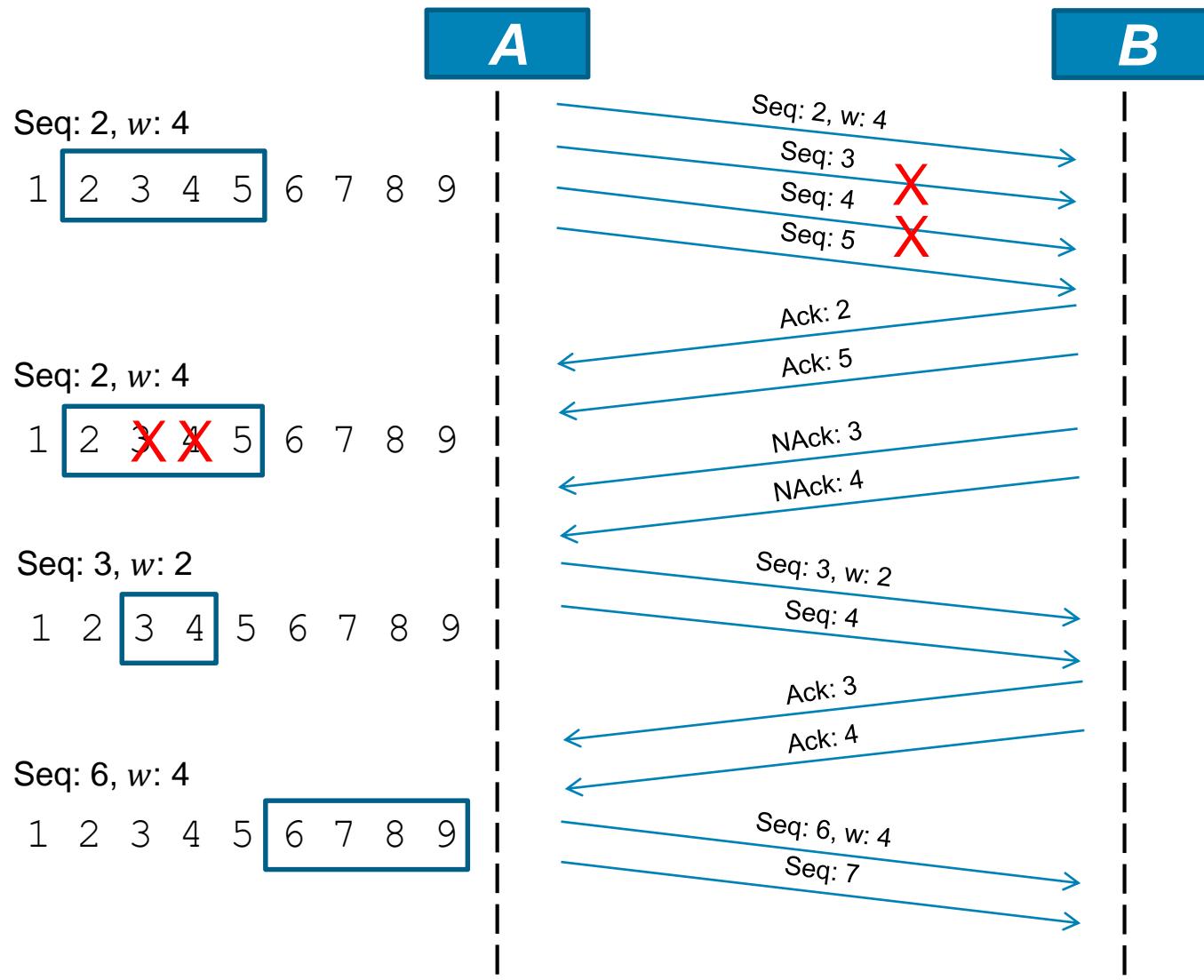
Packet speed to sender  
5000

Sender timeout  
50000

Sender timeout  
20000

- The receiver confirms all packets by bit mask after timeout or receive all packets in window
- Sender send all not acknowledged packet after timeout

# Selektivní znovu zasílání – NACK



# Selective Repeat – NACK: Demo

- <http://nes.fit.vutbr.cz/ivesely/pds/>

Go-Back-N   Selective Repeat - Fast Retransmit   Selective Repeat - Bit mask   **Selective Repeat - NACK**   SMART

PAUSE   SEND   RESTART

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

1 3

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

speed 1.00 + -

Send window size  
4

Packet speed to receiver  
5000

Packet speed to sender  
5000

Sender timeout  
50000

Receiver timeout  
20000

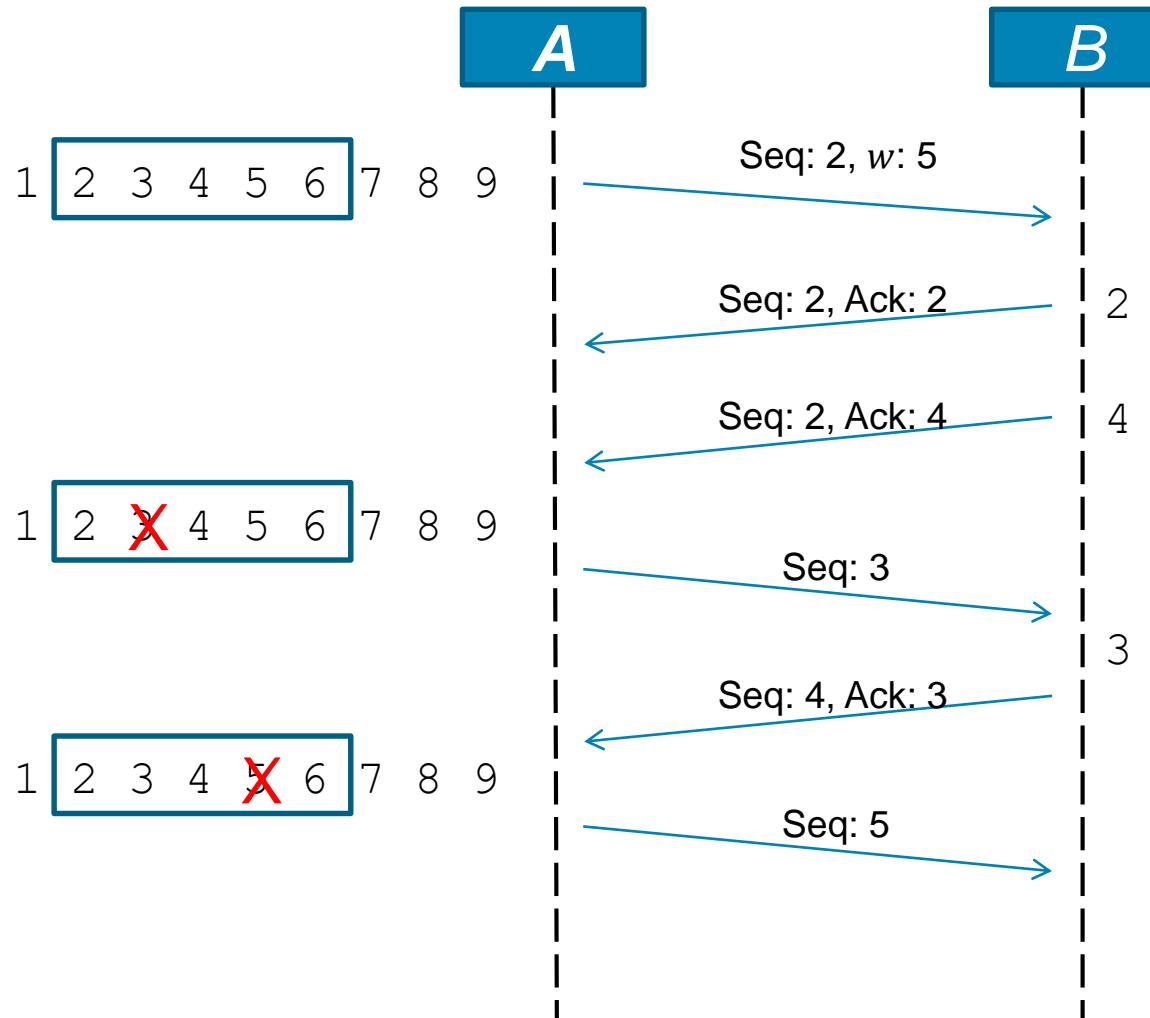
Packet  
Received  
Ack  
Ack received  
Nack

- The receiver confirms all packets independently.
- Sender send all not acknowledged packets after timeout
- Receiver send NACK packets after timeout

# SMART (1)

- Kombinace Selektivního znova zasílání s Go-Back-N a Fast Retransmit
- Princip:
  - Každý ACK obsahuje dvojici:  
*(poslední paket přijatý v pořadí, potvrzený paket),*  
která mu umožňuje zrekonstruovat bitovou mapu bez toho, aniž by ona samotná byla obsažena v hlavičce

# SMART (2)



# SMART ③

- Výhody:
  - oproti selektivnímu znovuzasílání nezvětšuje hlavičku
  - oproti Fast Retransmitu je schopno si poradit s více než jednou chybou v chyby-kontrolujícím okně
- Nevýhody:
  - nezbavuje bufferů na přeuspořádávání pořadí paketů

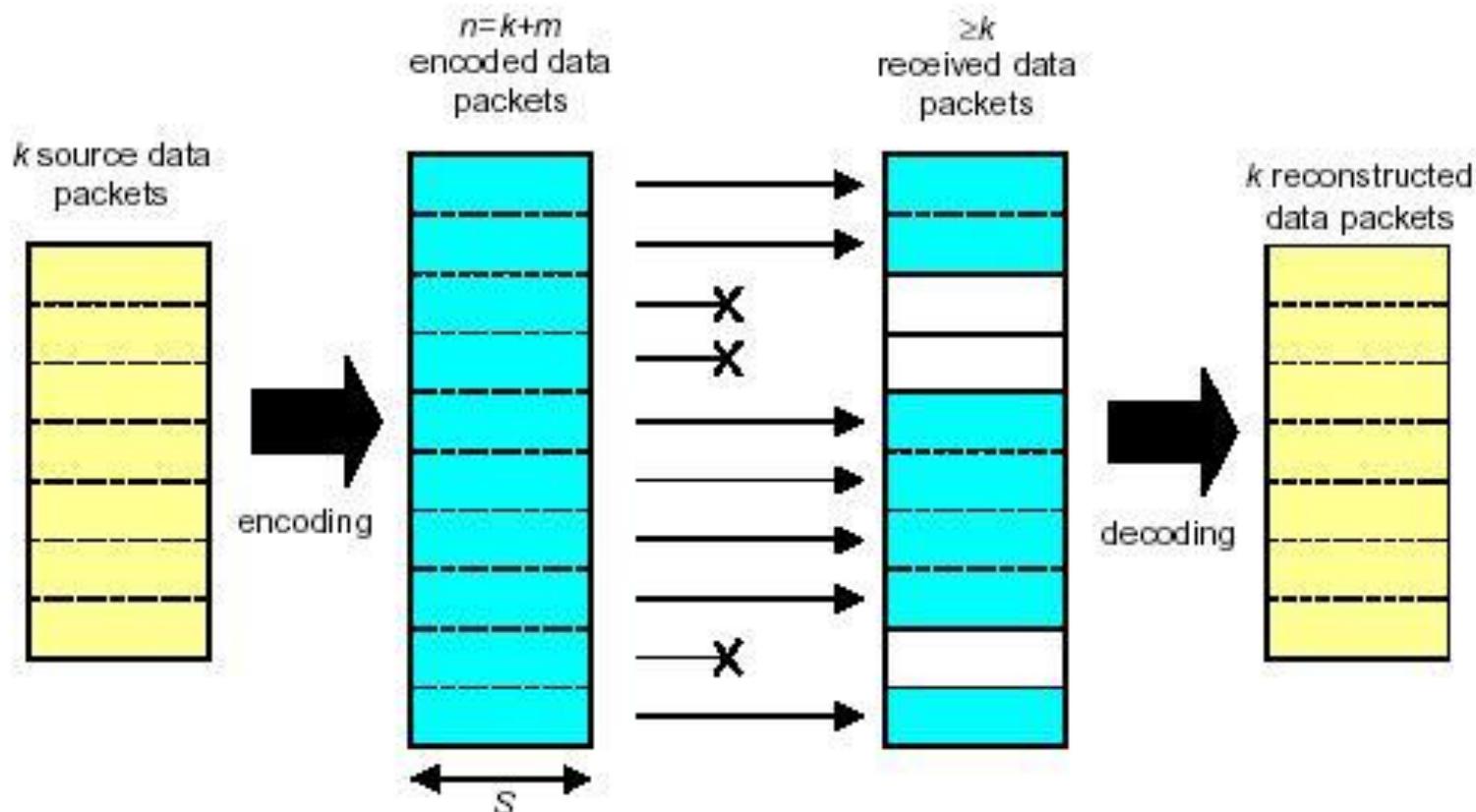
# Dodatky ke klouzavému oknu

- Jak na straně odesilatele, tak na straně příjemce
  - Každé může mít jinou délku
- *Jak velké ho nastavit?*
  - příliš malé → zvětšuje režii protokolu
  - příliš velké → zahlcení front middle-boxů
  - řešení = měnit ho podle chování sítě

# Forward Error Correction

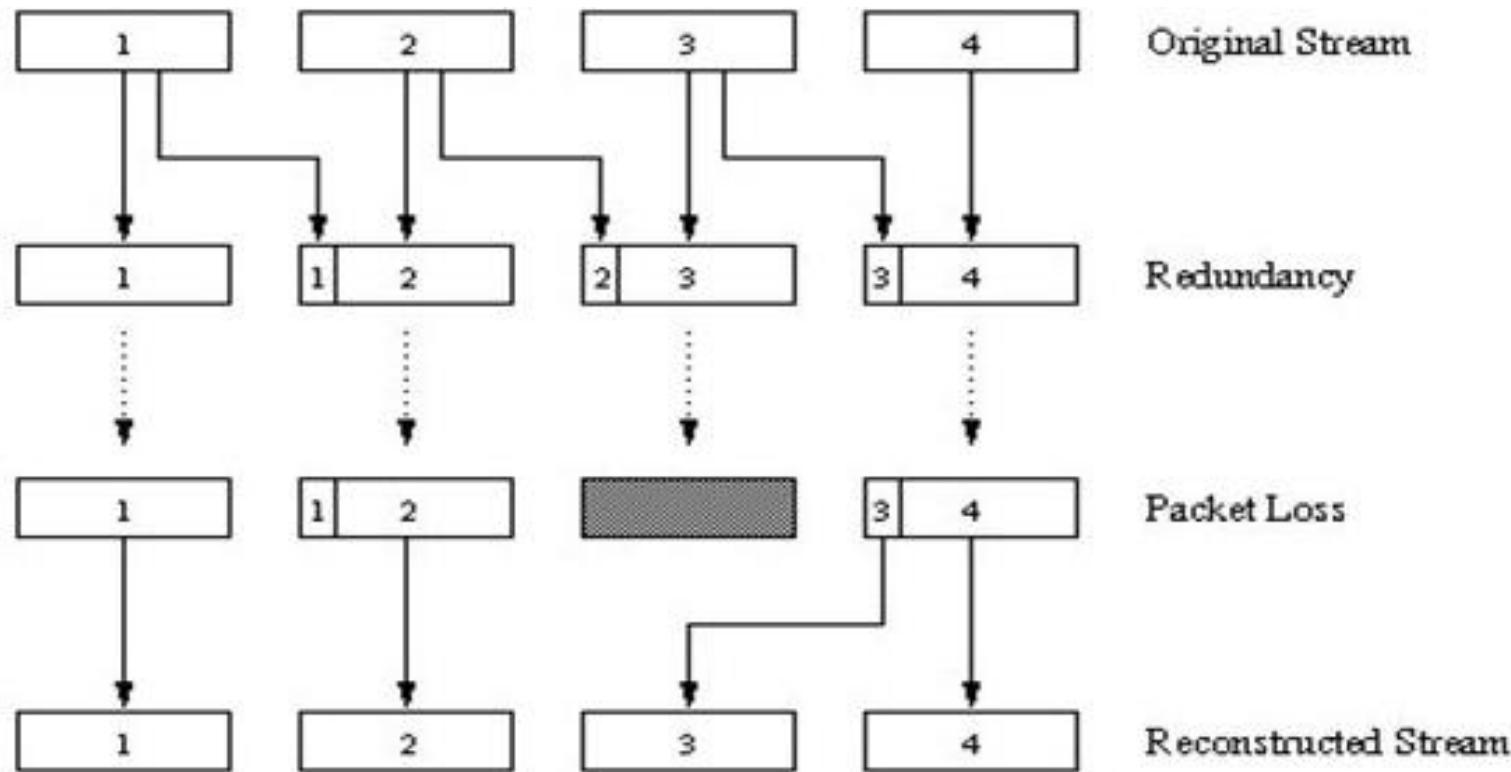
- Existují i jiné přístupy?
- FEC můžeme použít nejen na detekci a korekci bitových chyb, ale i na chyby paketů
- Princip:
  - Např. všechny pakety jsou stejné velikosti, přičemž každý osmý je paritním sedmi předchozích
- Výhody:
  - chyba jednoho paketu může být opravena bez znovuzaslání na straně příjemce
- Nevýhody:
  - vysoká režie na provoz
  - zvětšuje zpoždění (příjemce musí přijmout a ověřit celý FEC blok paketů)

# FEC - Ilustrace



[http://www.cs.technion.ac.il/Courses/Computer-Networks-Lab/projects/summer2001/SCTP/Final\\_report/sctp%20-%20final.htm](http://www.cs.technion.ac.il/Courses/Computer-Networks-Lab/projects/summer2001/SCTP/Final_report/sctp%20-%20final.htm)

# FEC - Ilustrace



# Zástupci

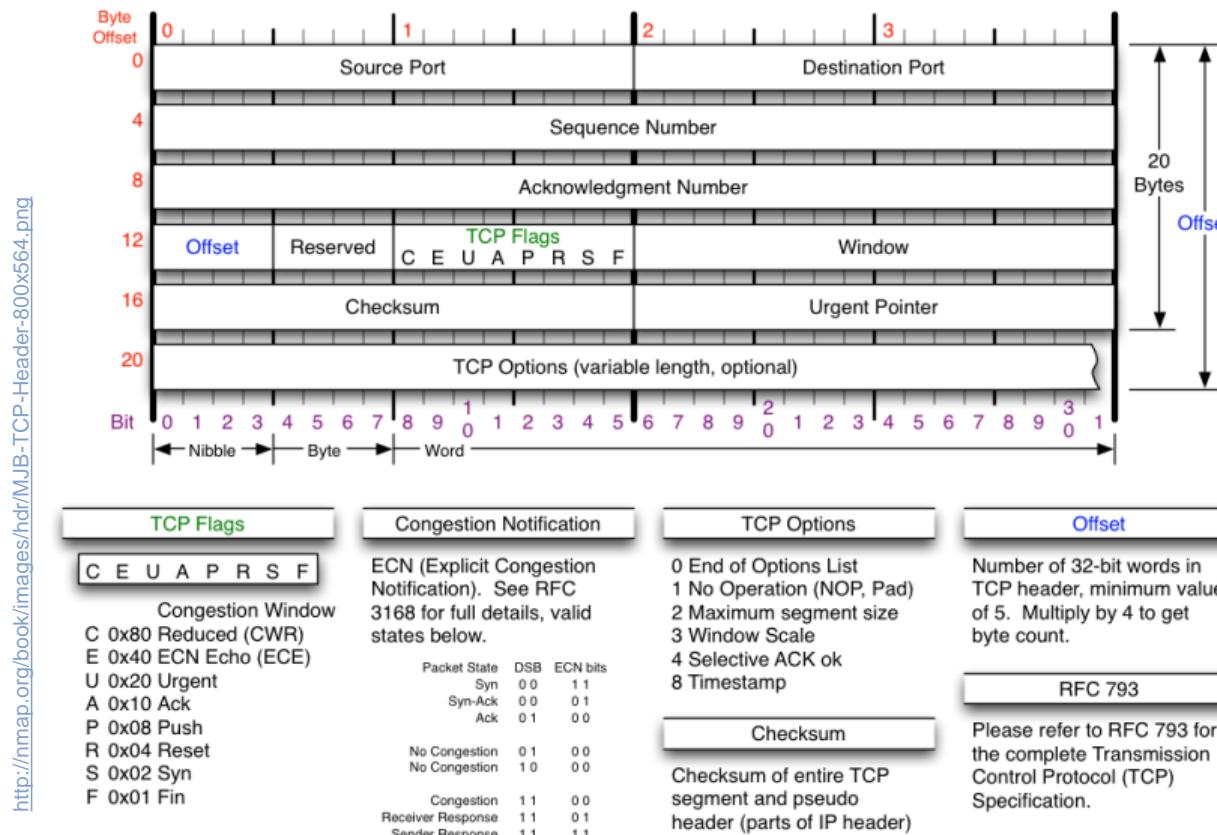
- *Jak vypadá teorie přetavená v praxi?*

# Transportní protokoly

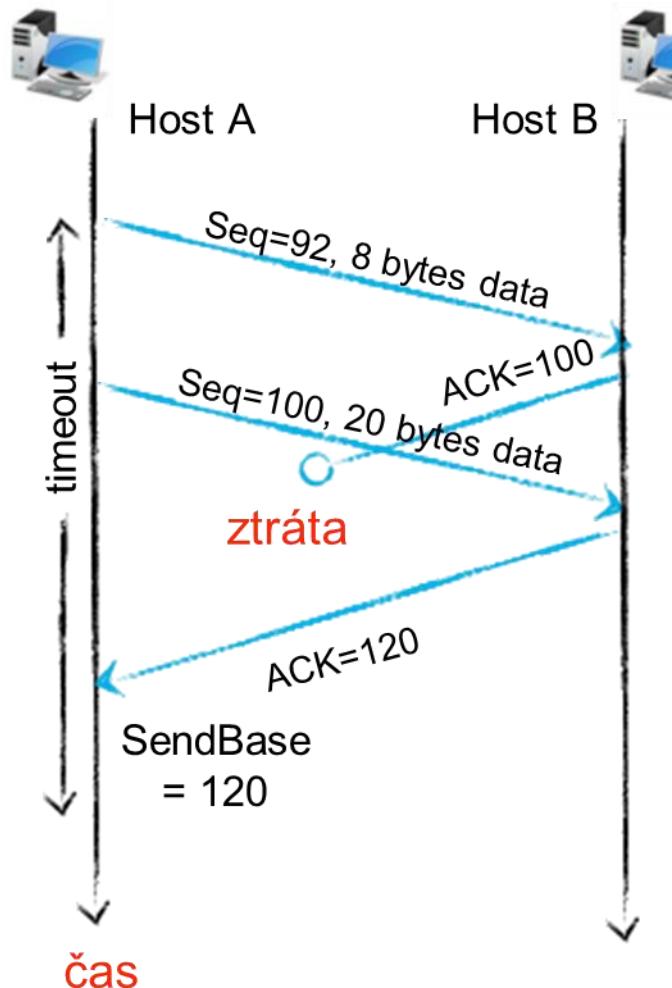
- ISO/OSI mantra transportní vrstvy:
  - Garantovat spolehlivé doručení (**reliable data transfer**)
  - Řídit tok (**flow control**)
  - Řídit zahlcení (**congestion control**)
  - Doručování v pořadí (**in-order delivery**)
    - Byte-order vs. Message-order
- Zástupci:
  - TCP
  - UDP
  - DCCP
  - SCTP
  - MPTCP
  - QUIC
  - Delta-t

# Transmission Control Protocol

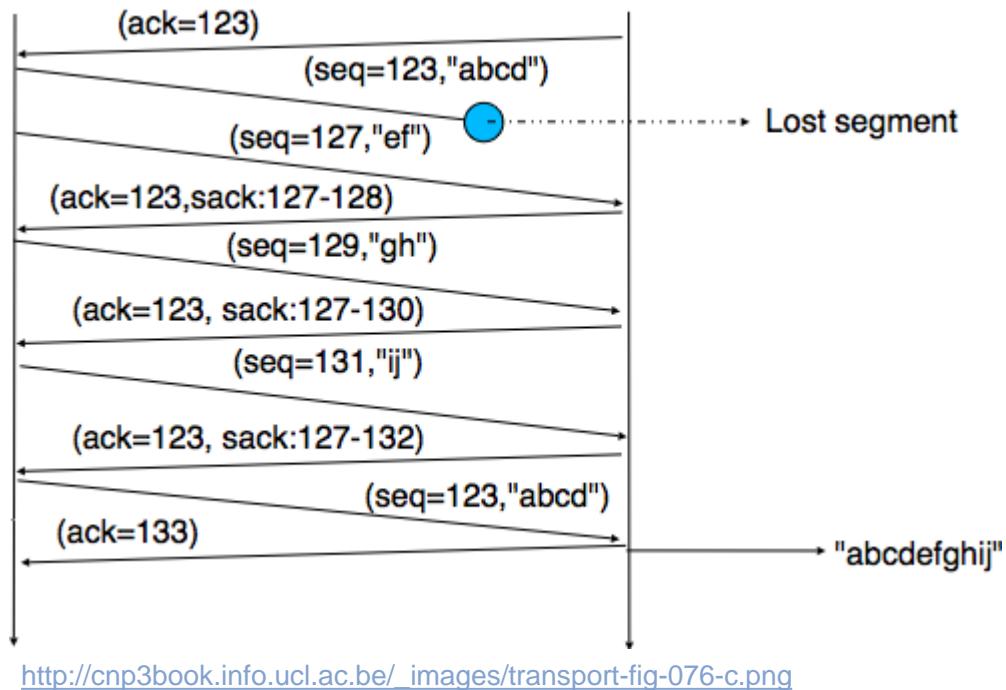
- RFC 793 z roku 1981
- Majorita aplikací využívá TCP
  - Provoz aplikace nad jiným transportním protokolem ☹



# Řízení toku – Potvrzení

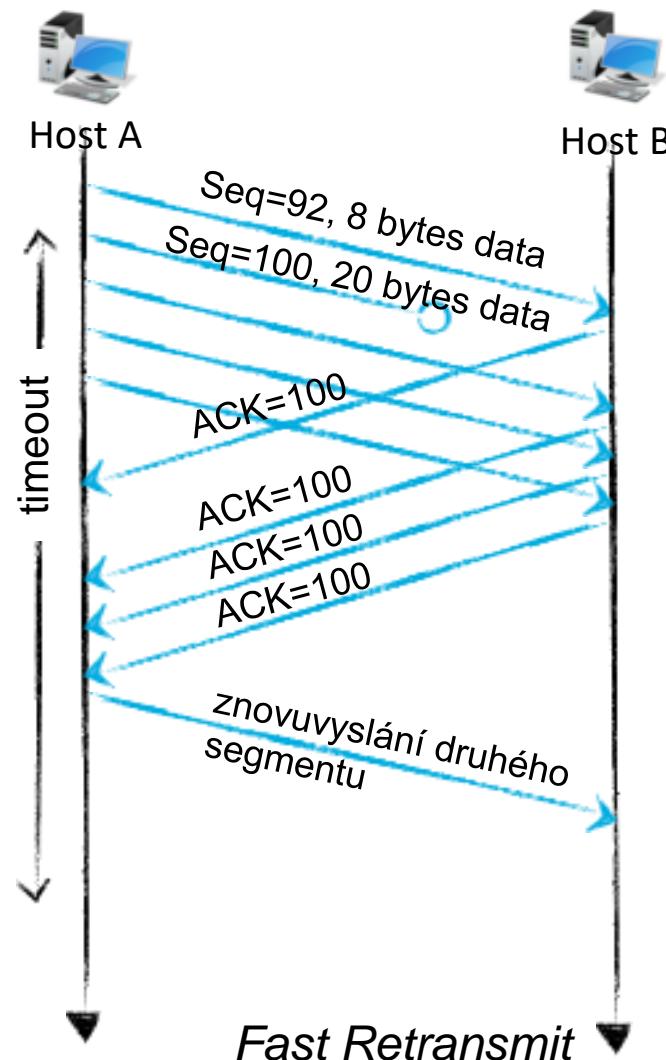


*Kumulativní*

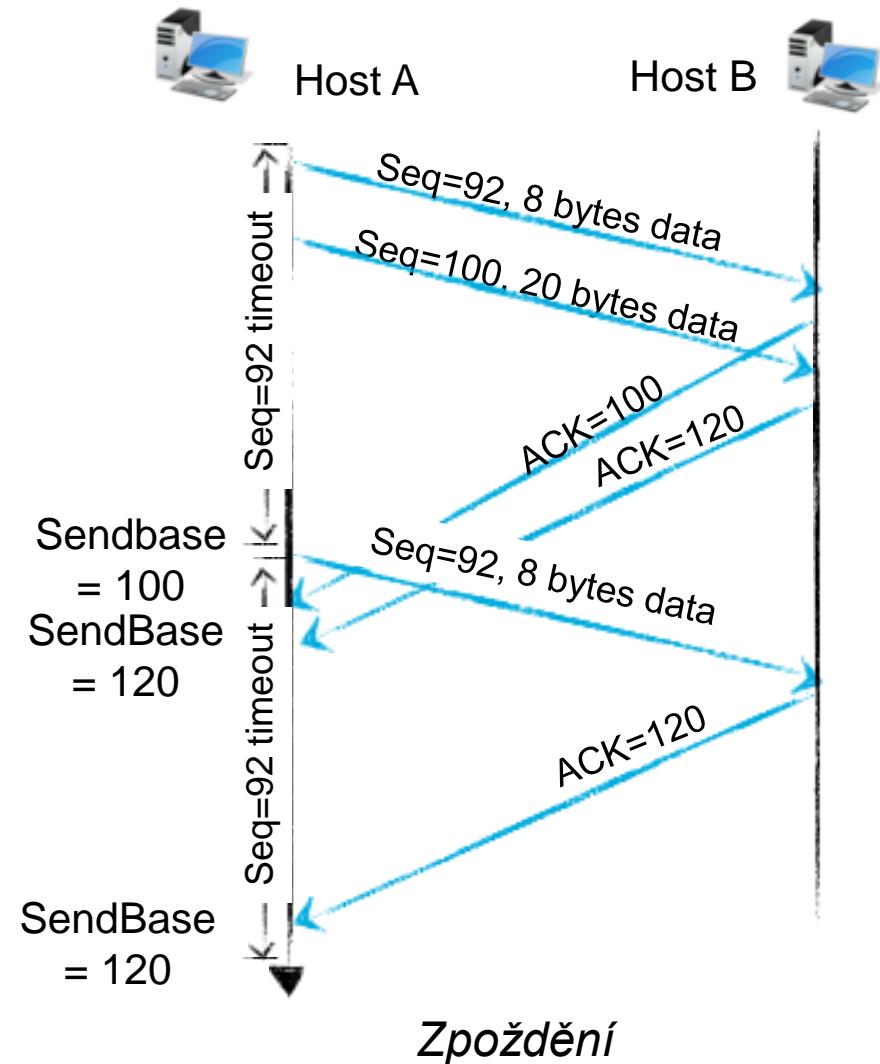
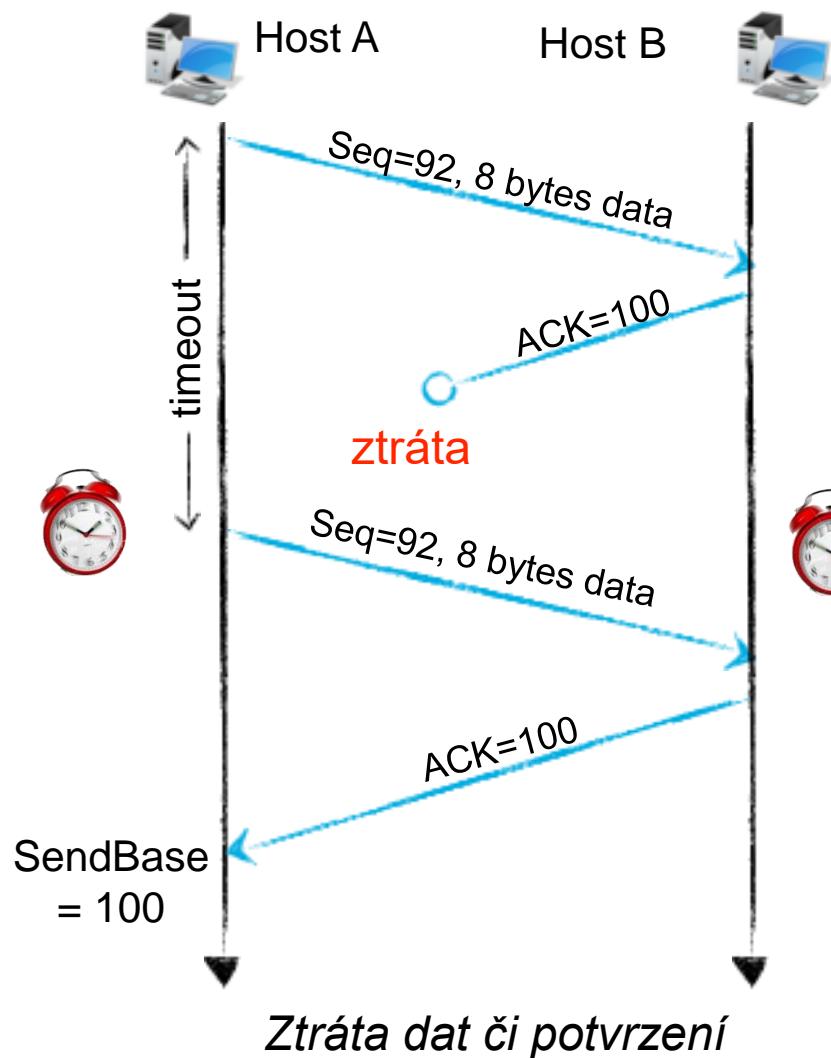


*Selektivní*

# Řízení toku – Znovuzasílání



# Řízení toku – Ztráty



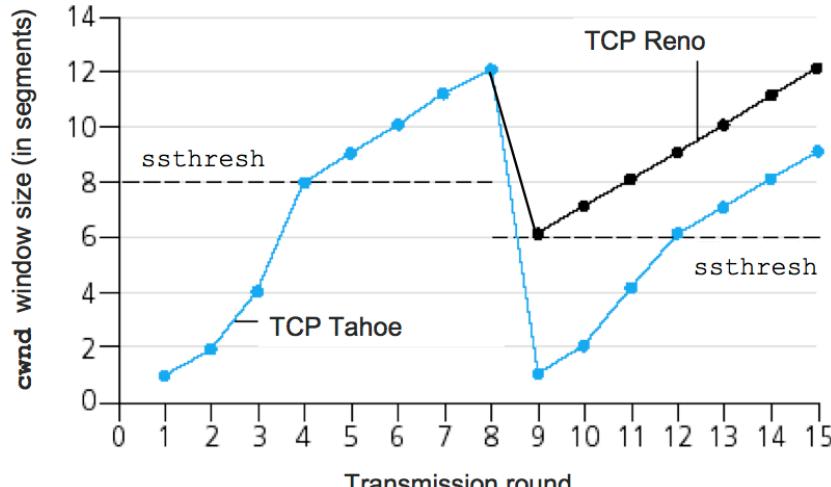
# TCP – Řízení zahlcení ①

- Historie
  - 1986: „First congestion collapse“
  - 1988: Van Jacobson a „Congestion Avoidance and Control“
- **Ztráta/zpoždění paketu = nastalo zahlcení!**
- Algoritmy:
  - TCP Tahoe
  - TCP Reno
  - TCP Vegas
  - TCP New Reno
  - TCP Hybla
  - TCP BIC
  - TCP CUBIC
  - Compound TCP
  - TCP Westwood, Westwood+
  - TCP SACK

# TCP – Řízení záhlcení

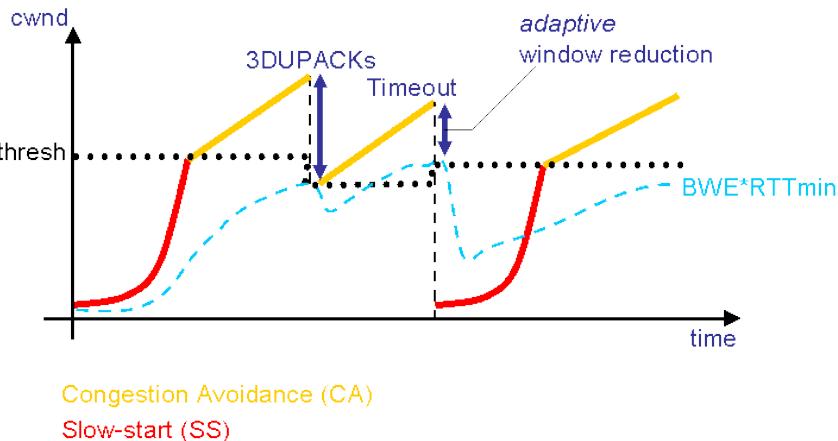
2

<https://gkf168.files.wordpress.com/2012/05/tahoe-reno.png>



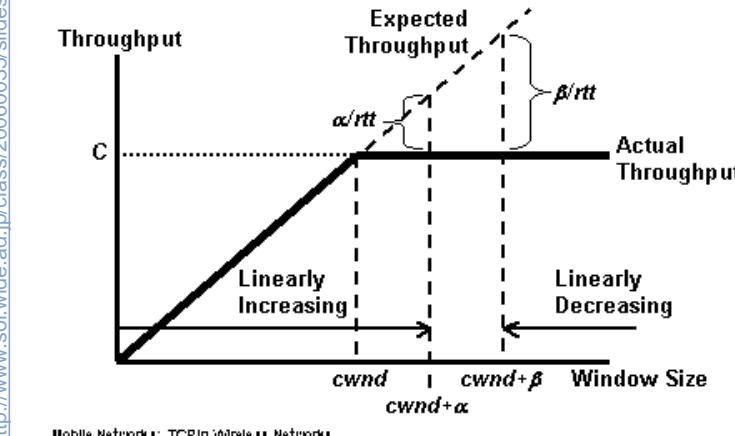
TCP Tahoe + Reno

[http://c3ab.poliba.it/images/5/5a/Tcp\\_westwood.gif](http://c3ab.poliba.it/images/5/5a/Tcp_westwood.gif)

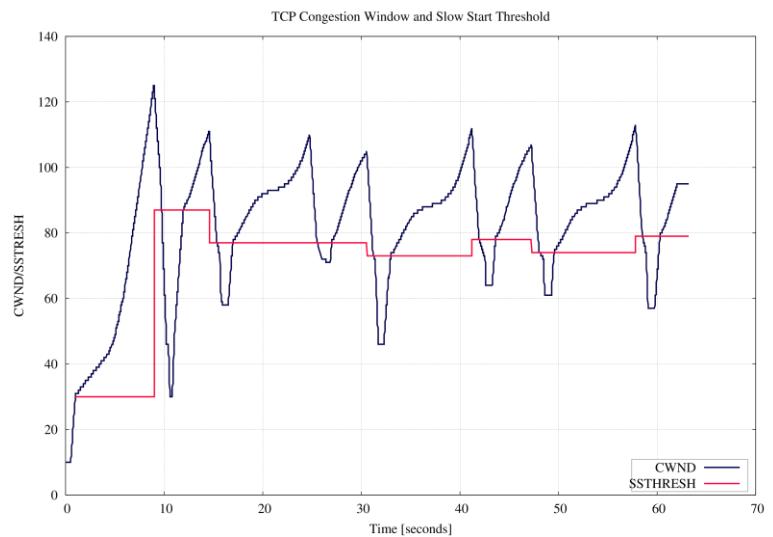


TCP Westwood

## Congestion Avoidance: TCP Vegas (3)



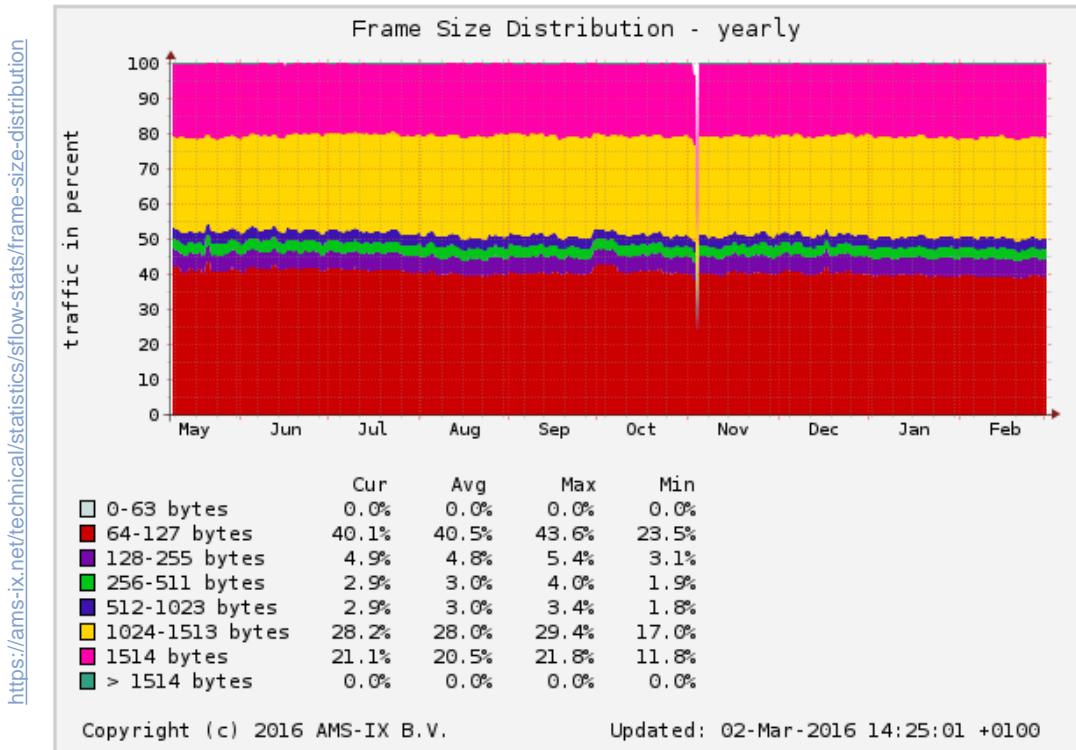
TCP Vegas



TCP CUBIC

# TCP – Nevýhody

- Head-of-line blocking
  - vše za ztraceným paketem je pozdrženo
- Nepodporuje aplikační multihoming
  - Pokud se změní IP adresa, musí se vytvořit nové TCP spojení
- Velká režie potvrzování

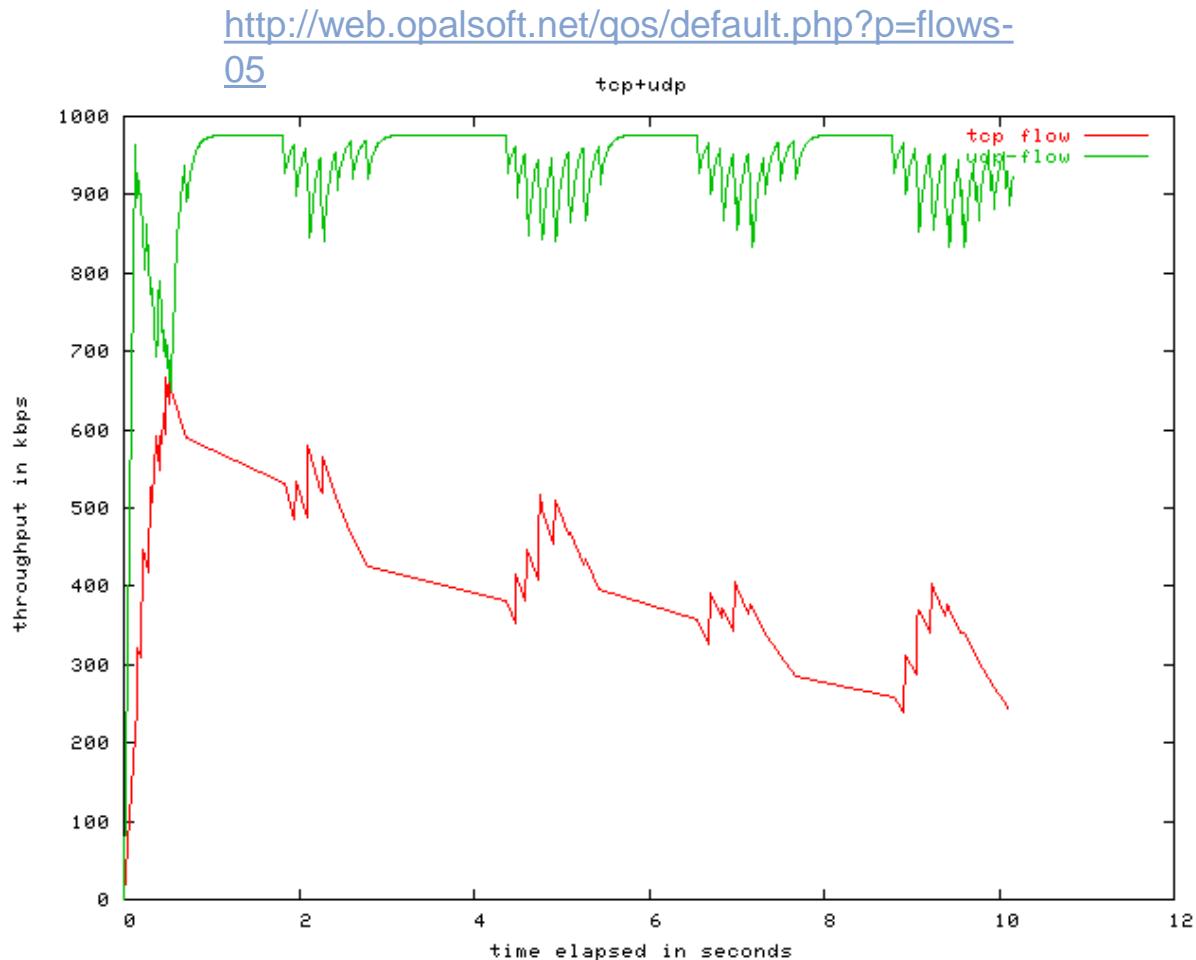


# TCP – Shrnutí

- Reliable transfer
  - Inorder byte stream
- Flow control
  - Fast Retransmit
- Congestion control
  - slow start + congestion avoidance
  - Spousta různých algoritmů
- Connection-oriented
  - 3-way handshake nastavující sekvenční čísla

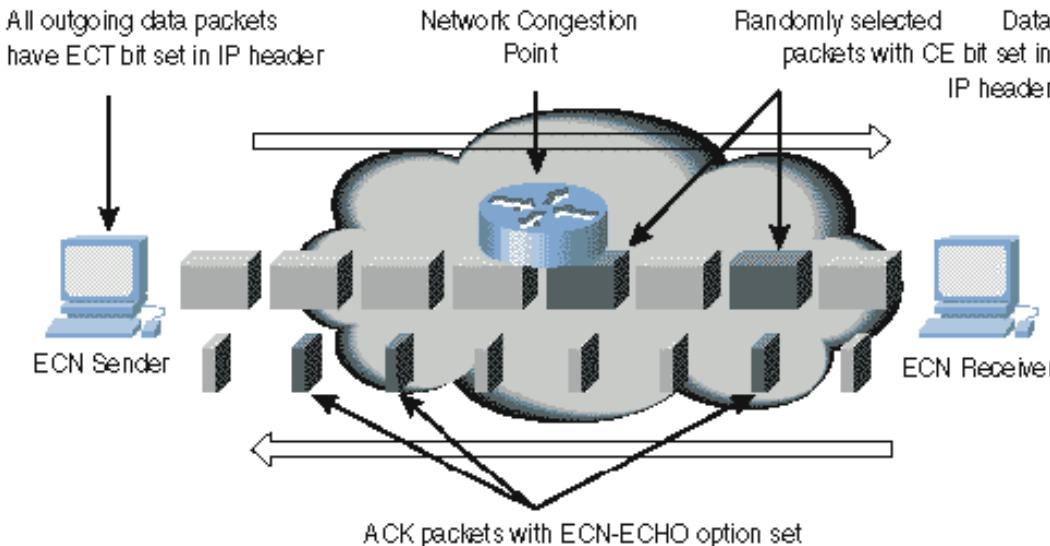
# User Datagram Protocol

- RFC 796  
z roku 1980
- Best-effort delivery  
+ Connection-less
  - *Spolehlivé doručování není moje zodpovědnost!*
  - *Tlačím data na plný plyn!*



# Datagram Congestion Control Protocol

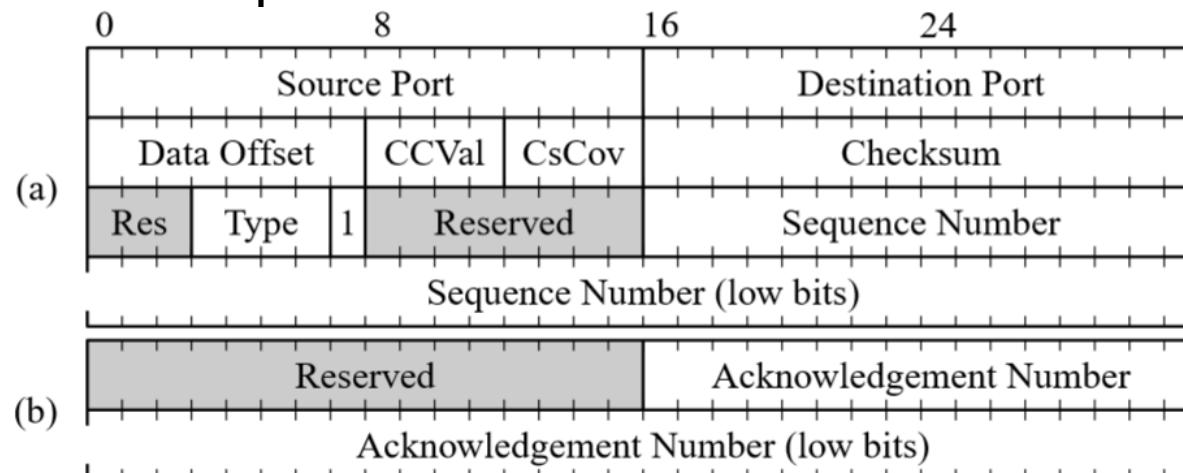
- [RFC 4336](#), [RFC 4340](#) a související
- Podpora řízení zahlcení pro UDP
  - Nastavování ECN bitu v potvrzeních



- Neposkytuje in-order doručování
  - Žádné znovuzasílání

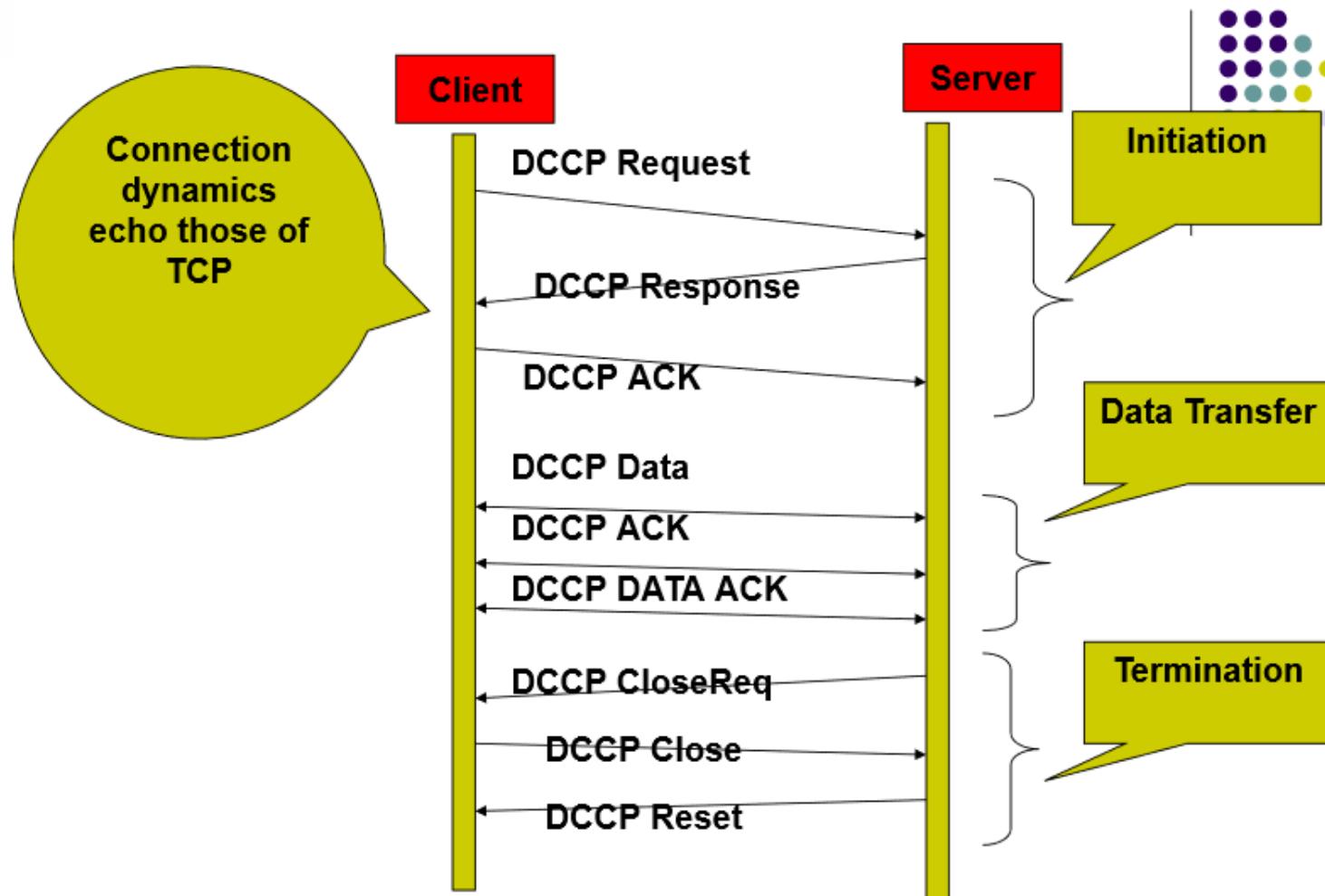
# DCCP – Zpráva

- Typy zpráv
  - Request-Response
  - Ack
  - Close-CloseReq-Reset



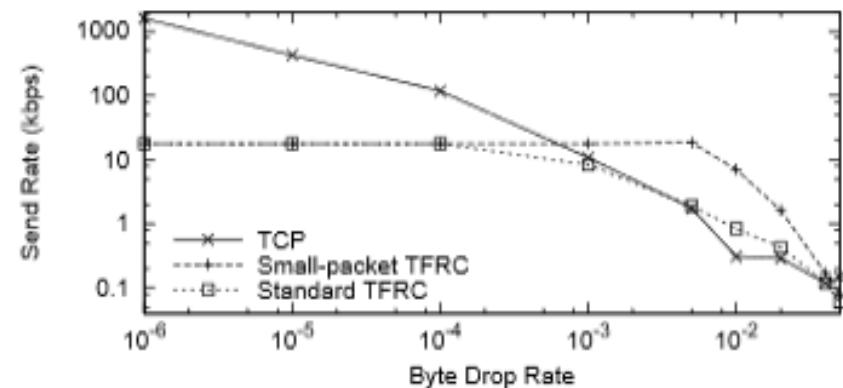
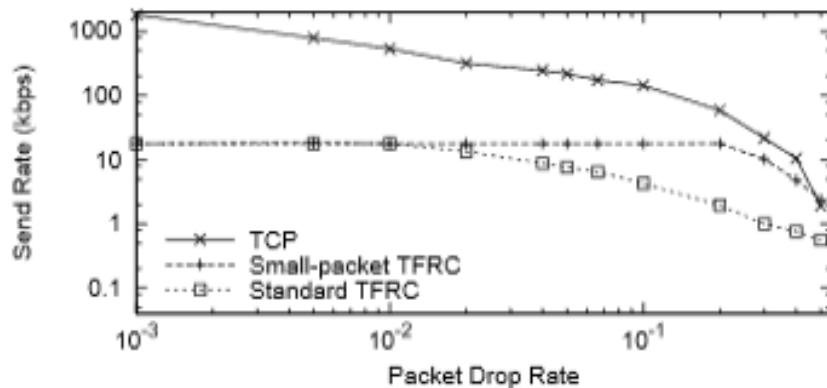
- 48bitová sekvenční čísla pro účtování DCCP, nikoli pro zajištění spolehlivého přenosu

# DCCP – Přenos



# DCCP – Řízení zahlcení

- Congestion Control ID v průběhu navazování spojení
- CCID2 = jako TCP
- CCID3 = jako Equation-based congestion control for unicast applications



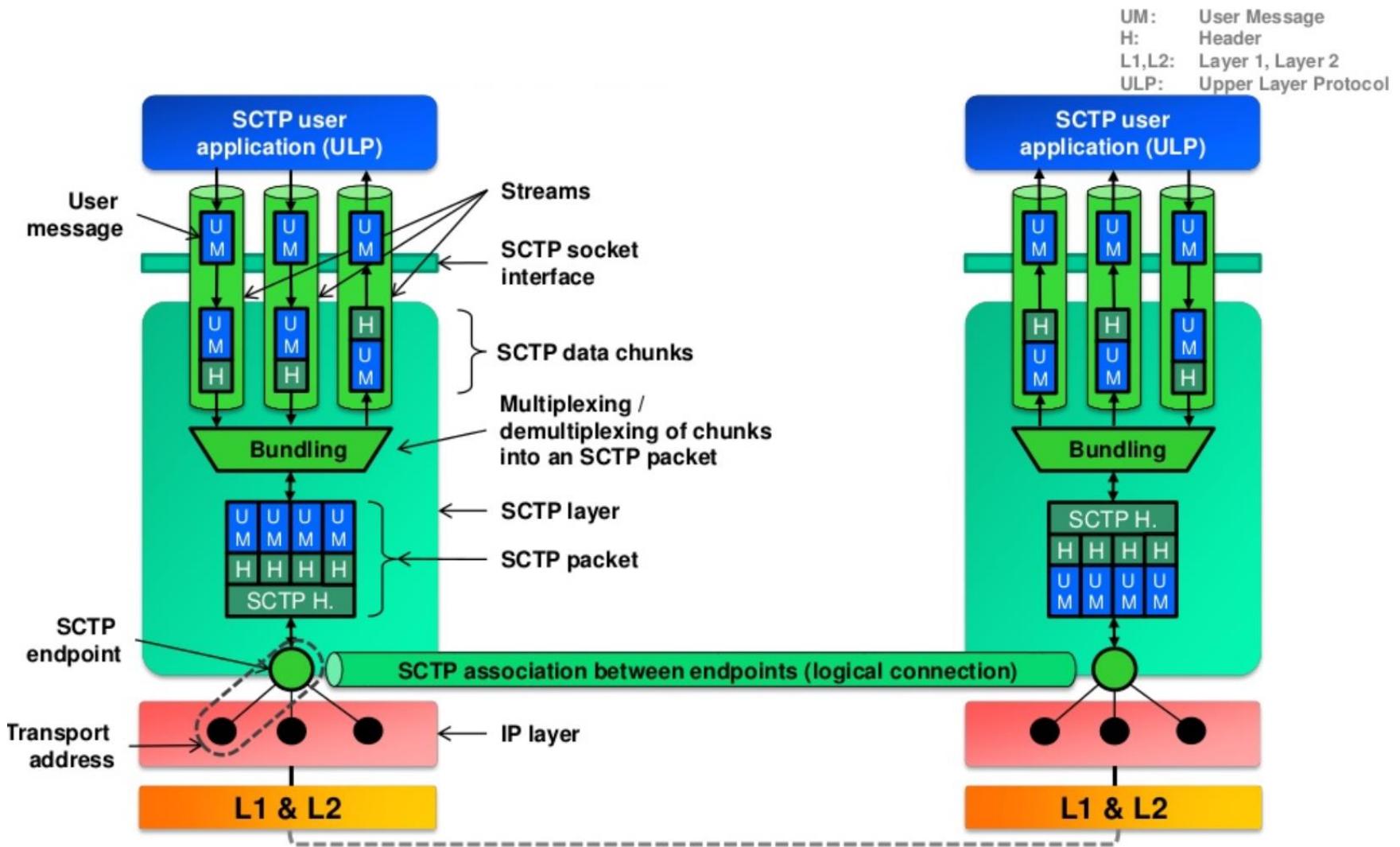
# DCCP – Shrnutí

- Best-effort delivery
  - Žádný flow control
- Congestion control
  - Bud' jako TCP
  - Vlastní algoritmy
- Connection-oriented
  - 3-way handshake domlouvající způsob řízení zahlcení

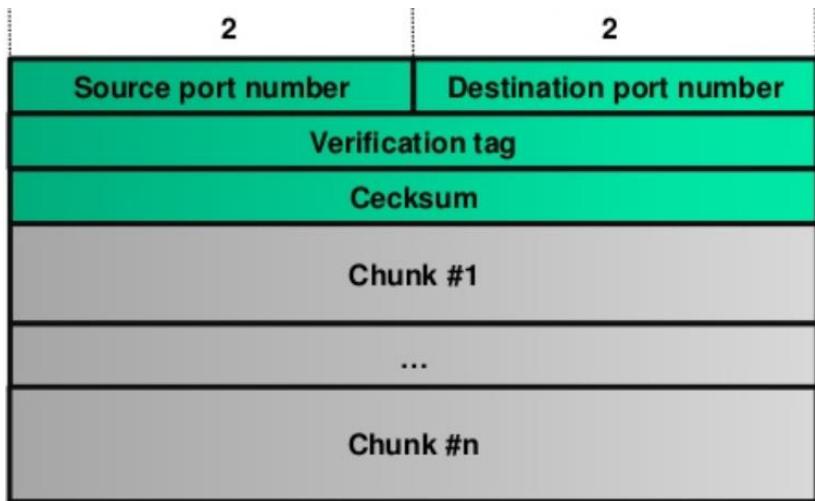
# Stream Control Transmission Protocol

- [RFC 3286](#), [RFC 4960](#) a související
- Vlastnosti
  - Bez HoL-blocking
    - Bundling zpráv (kombinace více zpráv do jedné SCTP)
  - Místo byte stream pracuje s message stream (hranice aplikacní dat jsou zohledňovány)
  - Relativně velká signalizační režie
- Connection management
  - Multihoming díky paralelním spojením skrz různé adresy
  - 4-way handshake (zabraňuje SYN-flood)
  - Path MTU discovery jako obrana proti fragmentaci

# SCTP – Ilustrace



# SCTP – Zpráva ①



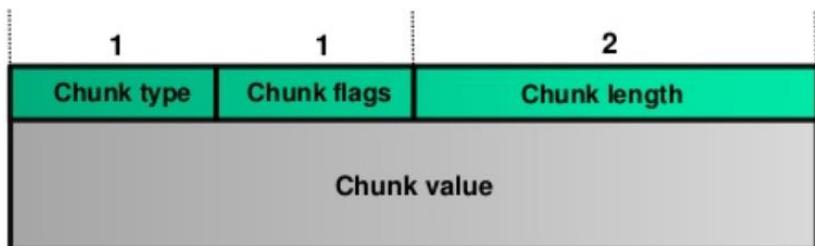
← Field length (bytes)

## SCTP common header:

- Source port #: Sender's port number  
Dest. port #: Receiver's port number  
Verif. tag: Used by receiver to validate the sender  
Checksum: CRC32 checksum over entire SCTP packet

## SCTP payload:

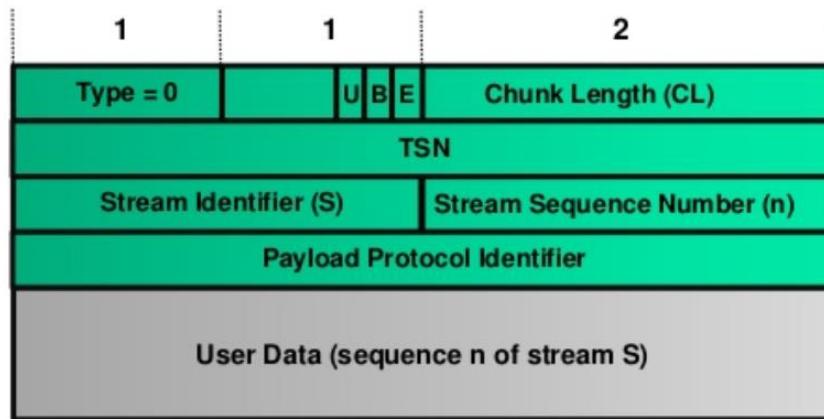
Sequence of user data and control chunks.  
Each chunk consists of a chunk header and a  
chunk value (chunk format see below).



## Chunk:

- Chunk type: DATA or control (INIT, INITACK etc.)  
Chunk flags: Chunk-specific bits  
Chunk length: Size of chunk including chunk header  
Chunk value: Chunk-specific data

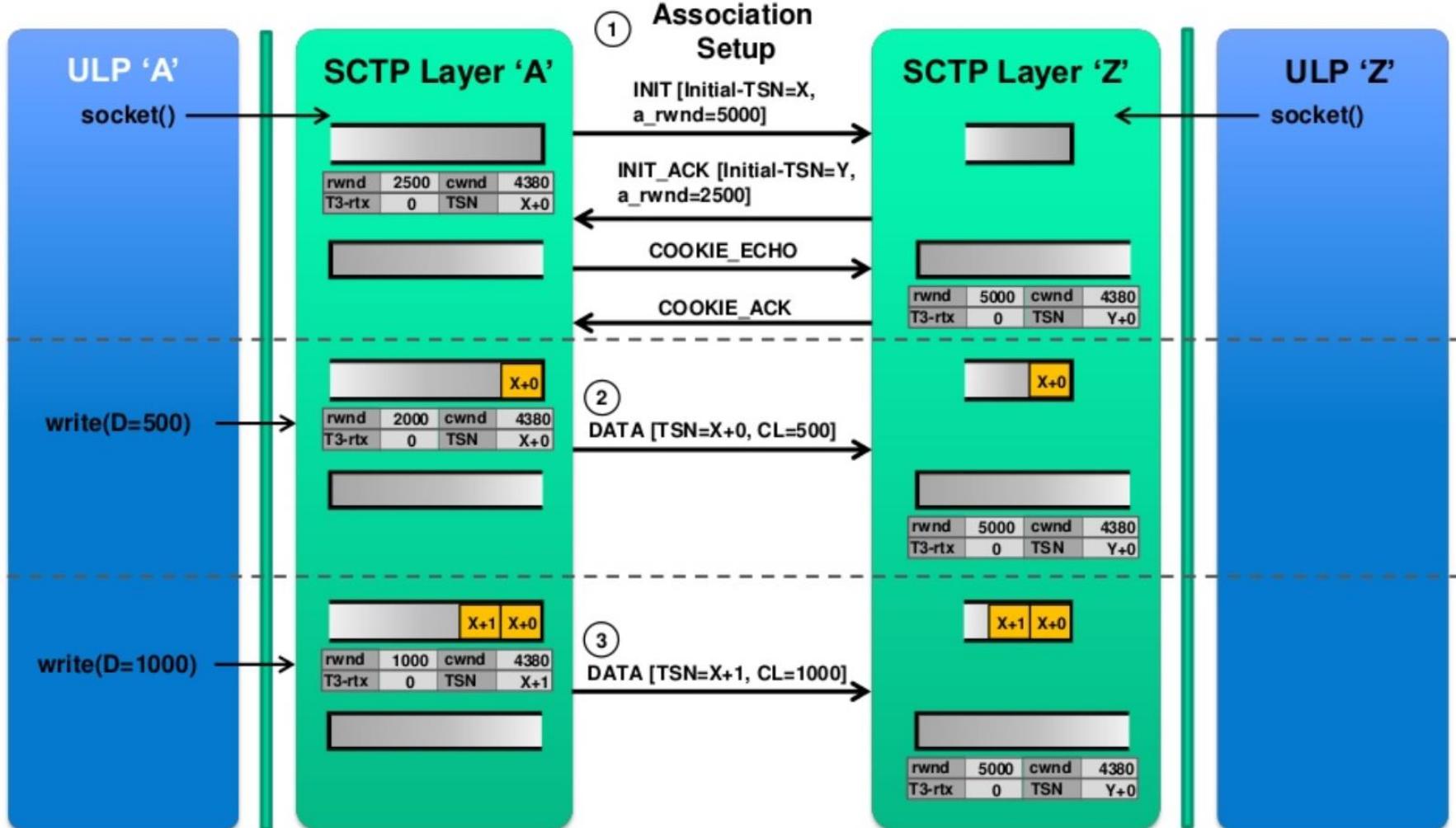
# SCTP – Zpráva ②



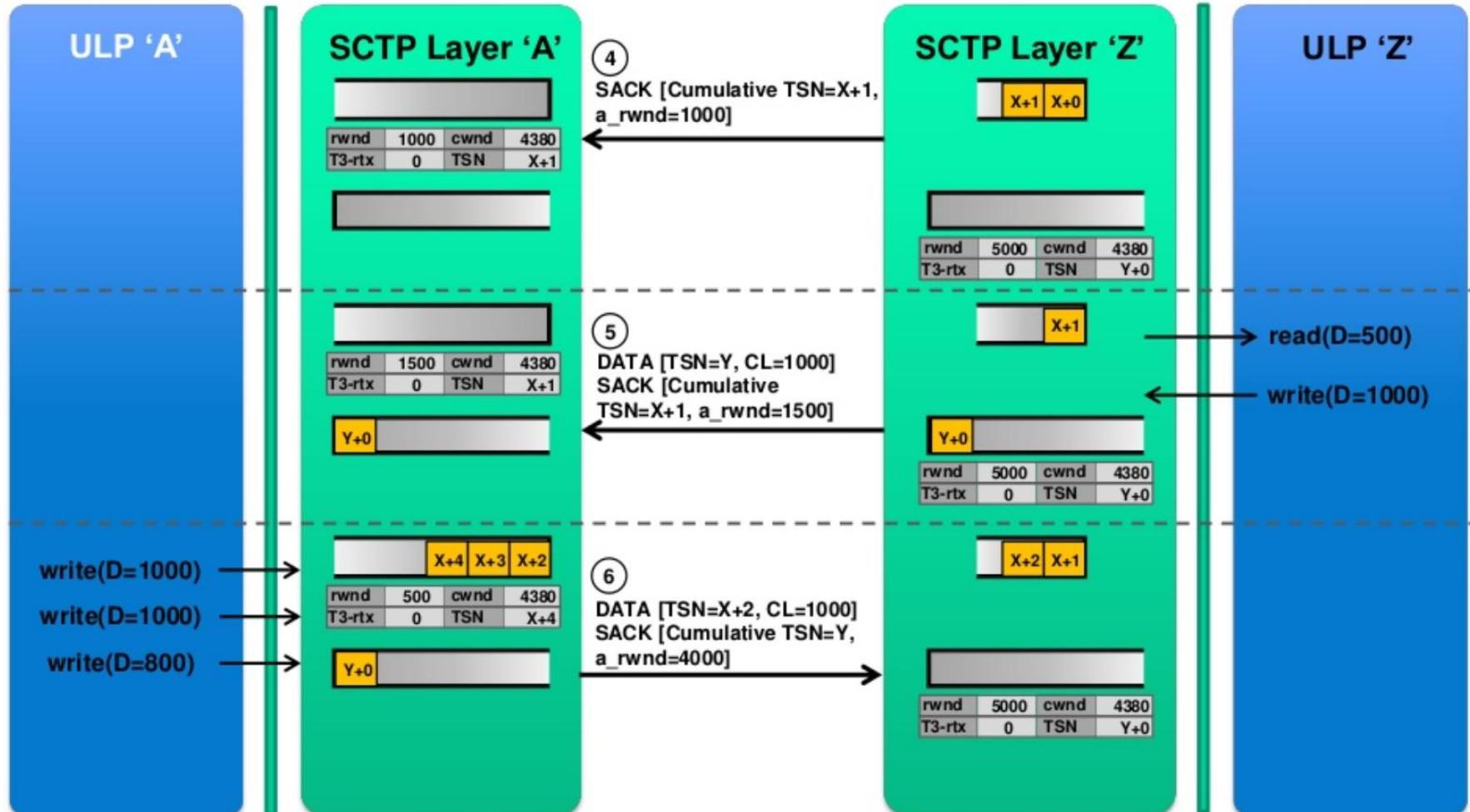
## Data chunk:

- U bit: If set to 1, indicates that this is an unordered chunk. Unordered chunks are transmitted and sent to the receiving application as is without re-ordering based on the sequence number.
- B bit: Begin of fragment bit. If set to 1 this is the first fragment of a larger fragmented user data message.
- E bit: End of fragment bit. If set to 1 this is the last fragment of a larger fragmented user data message.
- Stream identifier: Identifies the stream to which this chunk belongs.
- Stream seq. no.: Sequence number of user data within this stream. In case of fragmentation this number is identical for all fragments.
- Payload proto. id.: Identifies the upper (application) layer protocol (e.g. HTTP).
- User data: Application user data (e.g. HTTP header and payload).

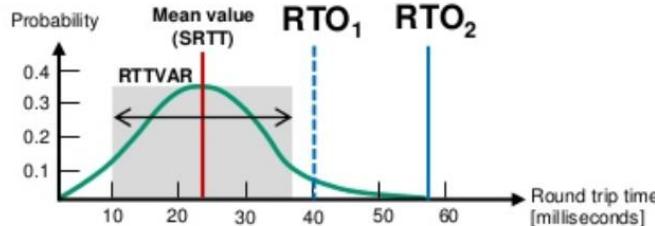
# SCTP – Přenos (1)



# SCTP – Přenos (2)



# SCTP – Retransmission Timeout



$RTO_1$  is too small. Many packets experience round trip times larger than  $RTO_1$  which would result in inadvertent retransmissions.  
 $RTO_2$  is a good choice since it is slightly larger than the largest round trip time but still as small as possible.

$$SRTT_{new} = (1 - RTO_\alpha) * SRTT_{old} + RTO_\alpha * R'$$

$$RTTVar_{new} = (1 - RTO_\beta) * RTTVar_{old} + RTO_\beta * |SRTT_{old} - R'|$$

$$RTO_{new} = SRTT_{new} + 4 * RTTVar_{new}$$

RTO is recalculated factoring in the currently smoothed («averaged») round trip time and its variation. The weighing factors ( $RTO_\alpha$  and  $RTO_\beta$ ) determine the speed of adjustment of RTO to new measurement values for the round trip time ( $R'$ ).

where

$RTTVar$  = current Round Trip Time variation

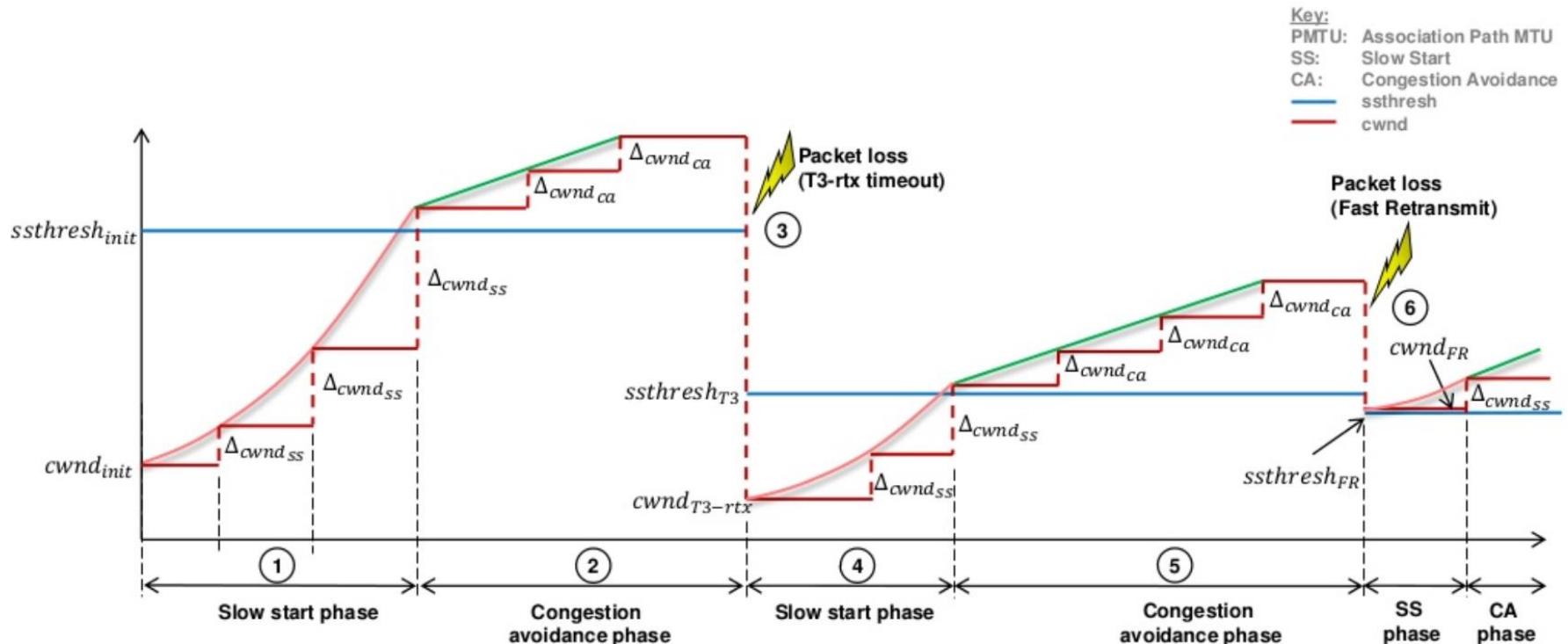
$SRTT$  = current Smoothed Round Trip Time

$RTO$  = current Retransmission TimeOut timer value

$RTO_\alpha$  = weighing factor (default value = 1/8)

$RTO_\beta$  = weighing factor (default value = 1/4)

# SCTP – Řízení zahlcení



$$cwnd_{init} = \min(4 * PMTU, \max(2 * PMTU, 4380 \text{ bytes}))$$

$ssthresh_{init} = \infty$  (or  $a\_rwnd$ )

$$\Delta cwnd_{ss} = \min(DATA_{acked}, PMTU)$$

$\Delta cwnd_{ca} = 1 * PMTU \text{ per RTT}$

$$cwnd_{T3\text{-rtx}} = 1 * PMTU$$

$$cwnd_{FR} = ssthresh_{FR}$$

$$ssthresh_{T3} = ssthresh_{FR} = \max\left(\frac{cwnd}{2}, 4 * PMTU\right)$$

$burst_{max} = 4$

# SCTP ve Wireshark'u

The screenshot shows the Wireshark interface with the following details:

- Filter:** sctp
- Packets:** 396
- Displayed:** 40
- Marked:** 0
- Dropped:** 0

**Selected Packet Details:**

No.	Time	Source	Destination	Protocol	Info
214	134.943324	127.0.0.1	127.0.0.1	SCTP	INIT
215	134.943462	127.0.0.1	127.0.0.1	SCTP	INIT_ACK
216	134.943530	127.0.0.1	127.0.0.1	SCTP	COOKIE_ECHO DATA
217	134.943648	127.0.0.1	127.0.0.1	SCTP	COOKIE_ACK SACK
221	137.013151	192.168.0.20	192.168.0.20	SCTP	HEARTBEAT

**Packet 216 Analysis:**

- Frame 216: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits)
- Ethernet II, Src: 00:00:00\_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00\_00:00:00 (00:00:00:00:00:00)
- Internet Protocol, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
- Stream Control Transmission Protocol, Src Port: 260 (260), Dst Port: 250 (250)
  - Source port: 260
  - Destination port: 250
  - Verification tag: 0x03143acd
  - Checksum: 0x00000000 (not verified)
- COOKIE\_ECHO chunk (Cookie length: 260 bytes)
- DATA chunk(ordered, complete segment, TSN: 238345625, SID: 0, SSN: 0, PPID: 0, payload length: 15 bytes)

**Selected Data:** Data (15 bytes)

0110 00 00 80 00 00 04 c0 00 00 04 00 05 00 08 7f 00	.....
0120 00 01 00 05 00 08 c0 a8 00 14 00 00 00 00 00 00	.....
0130 00 00 00 00 00 00 00 03 00 1f 0e 34 dd 99 00 00	..... 4....
0140 00 00 00 00 00 00 48 65 6c 6c 6f 20 53 43 54 50	..... Hello SCTP
0150 21 21 21 0a 00 00	!!!..

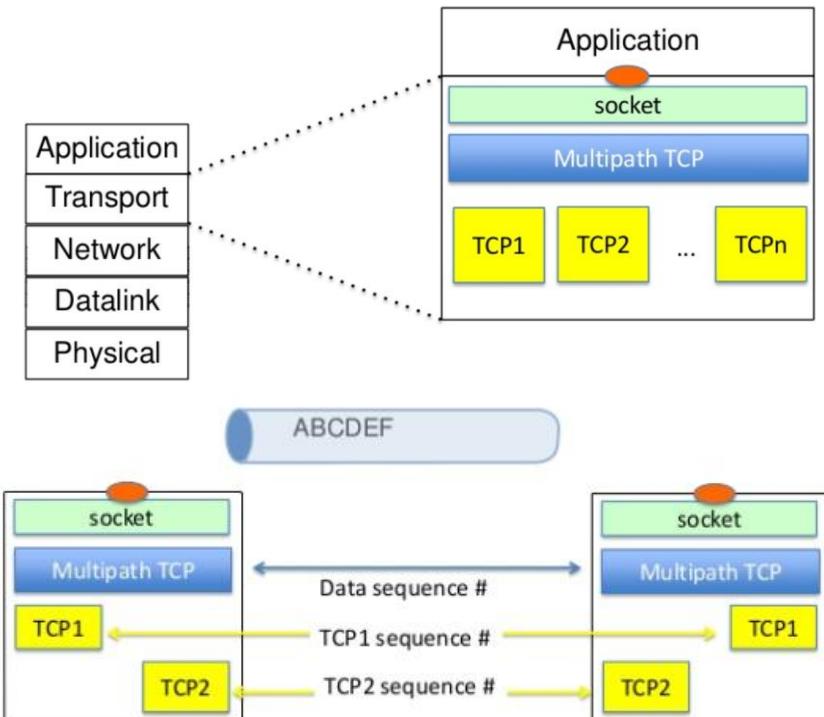
http://3.bp.blogspot.com/-FcPzyOcvGw/The20r1HU2i/AAAAAAAABoA/c56RfNkfzci/s1600/05\_wireshark\_sctp\_data.png

# SCTP – Shrnutí

- Reliable transfer
  - Inorder message stream
- Flow control
  - EWMA
  - Fast Retransmit
- Congestion control
  - slow start + congestion avoidance
- Connection-oriented
  - 4-way handshake nastavující sekvenční čísla a bránící před SYN floodem

# Multipath TCP

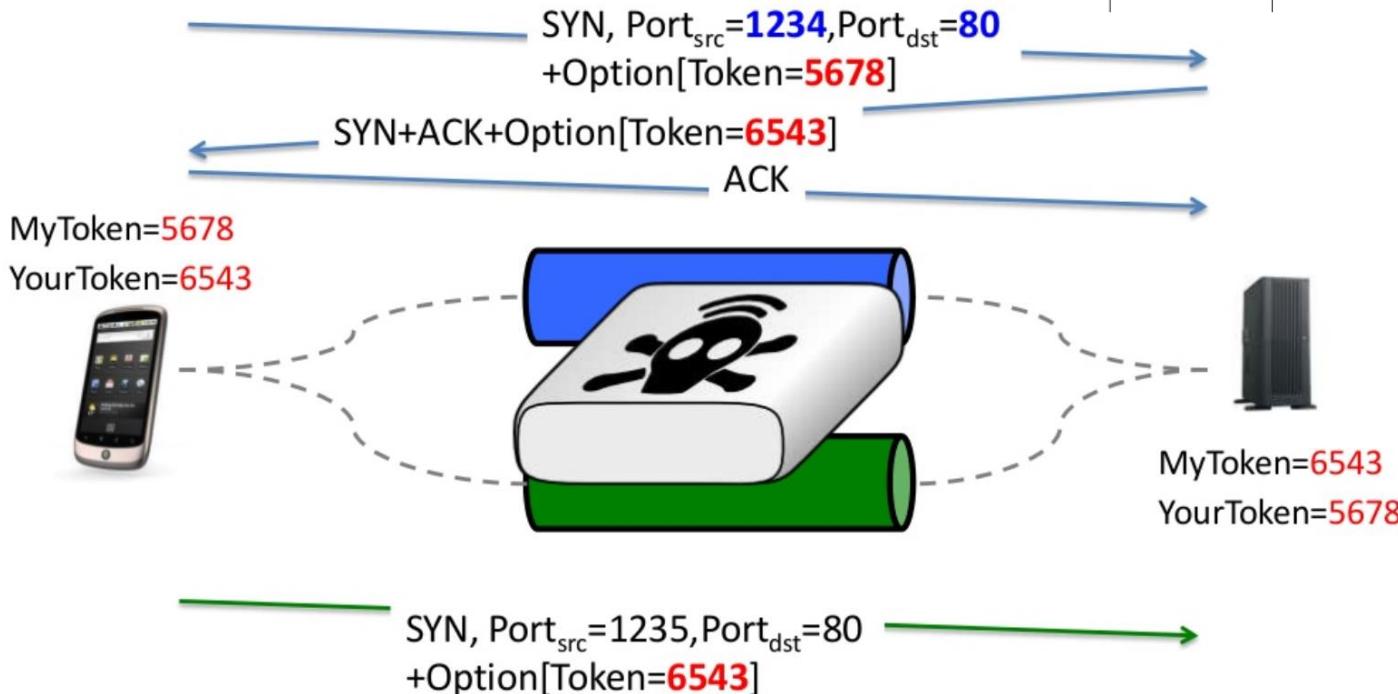
- [RFC 6824](#) a související
- Multiplexing TCP skrz více spojení
  - Obdobný socketový interface jako TCP
  - Menší signalizační náročnost oproti SCTP



Ver	IHL	ToS	Total length		
Identification		Flags	Frag. Offset		
TTL	Protocol	Checksum			
Source IP address					
Destination IP address					
Source port		Destination port			
Sequence number					
Acknowledgment number					
IHL	Reserved	Flags	Window		
Checksum		Urgent pointer			
Options					
Payload					

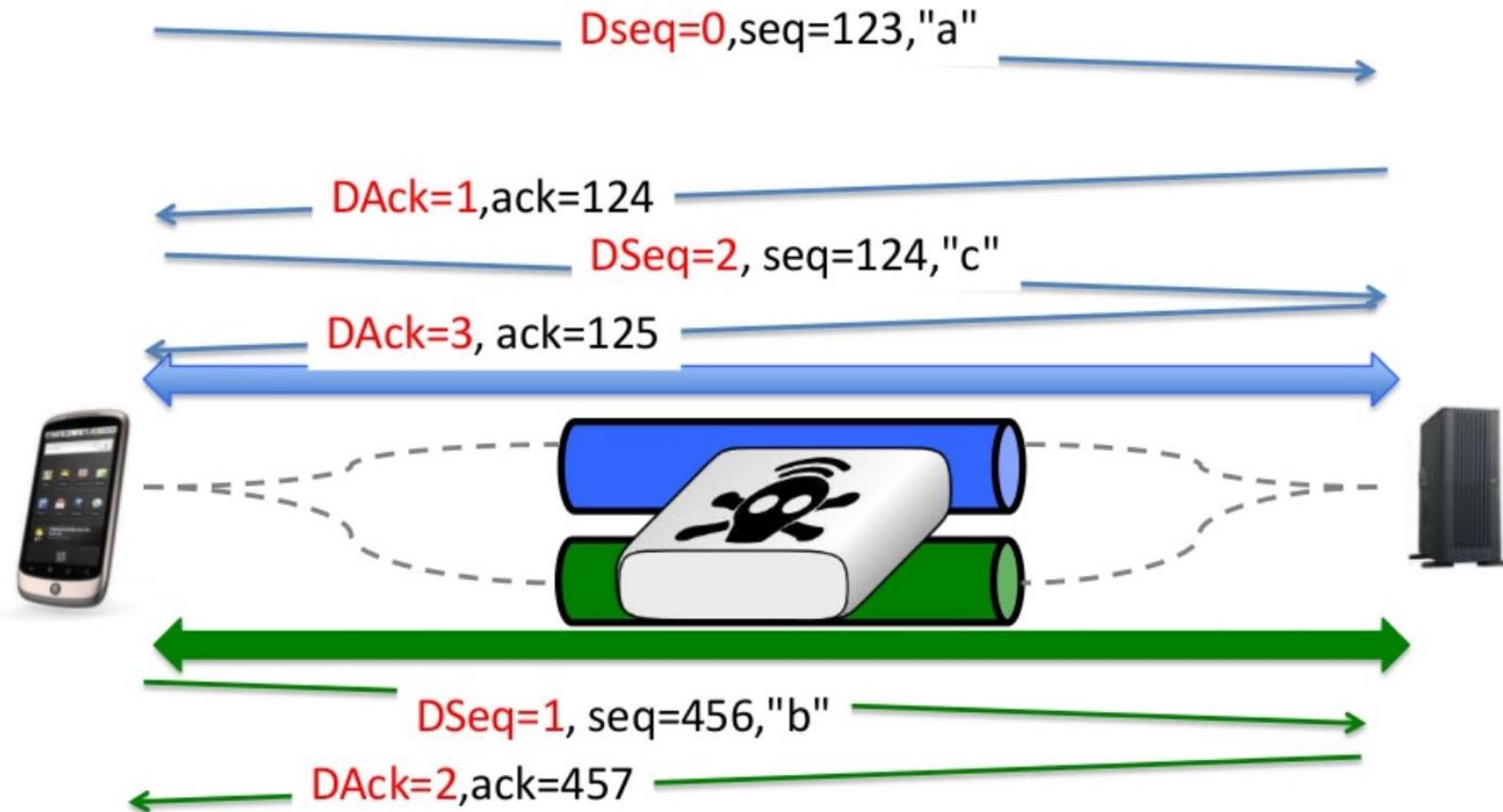
# MPTCP – Řízení spojení

- Při navázání spojení posílá dodatečnou Option



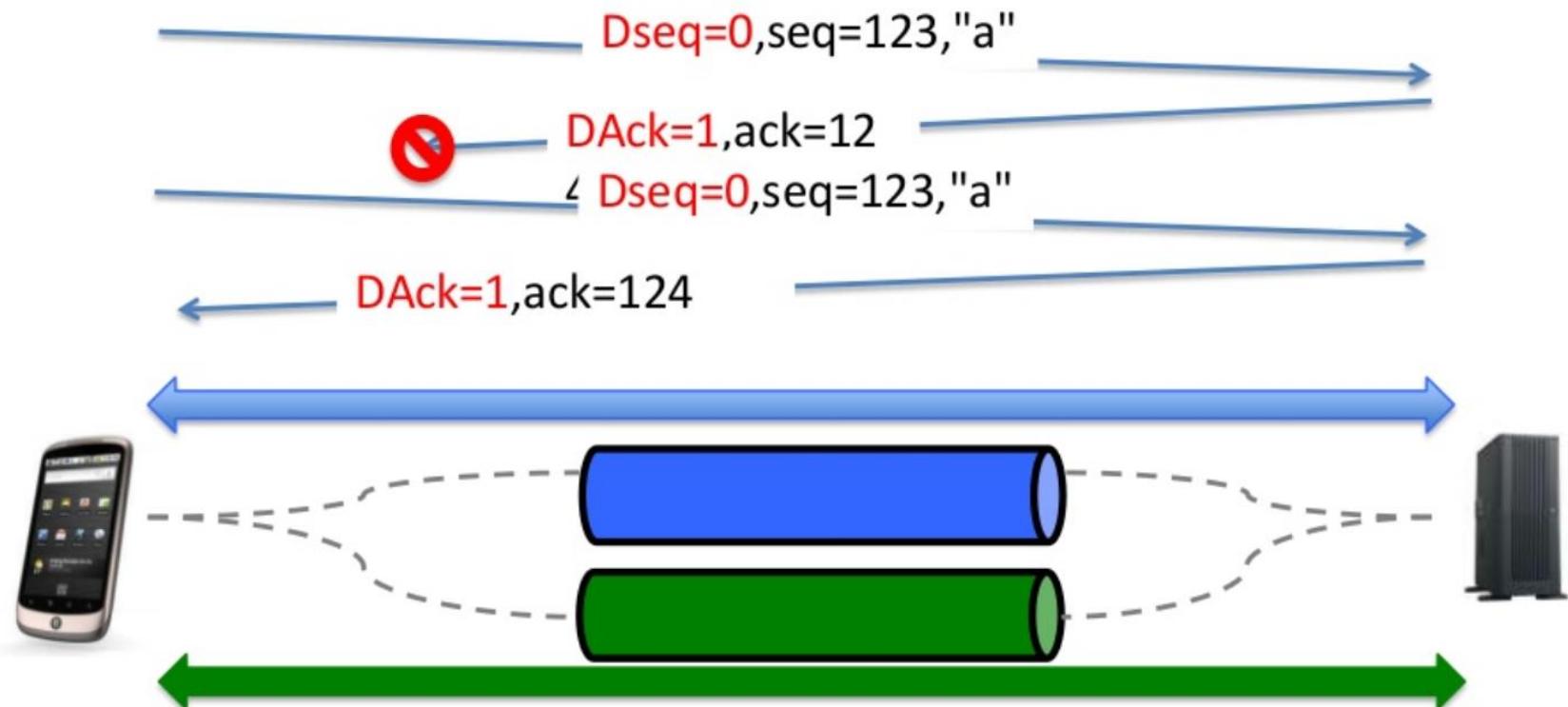
# MPTCP – Řízení toku ①

- Dva druhy sekvenčních čísel – subflow a data



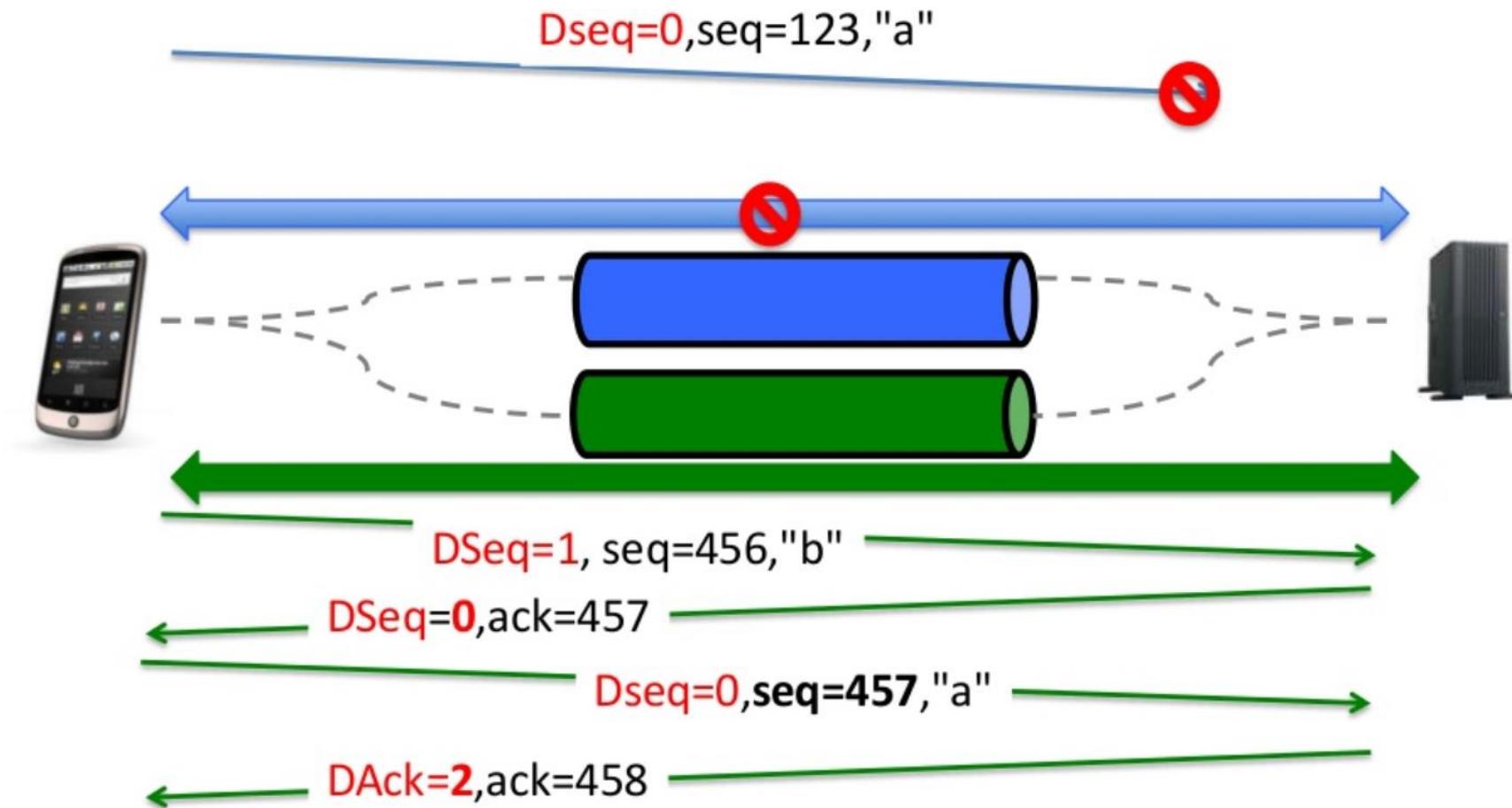
# MPTCP – Řízení toku ②

- Znovuzaslání



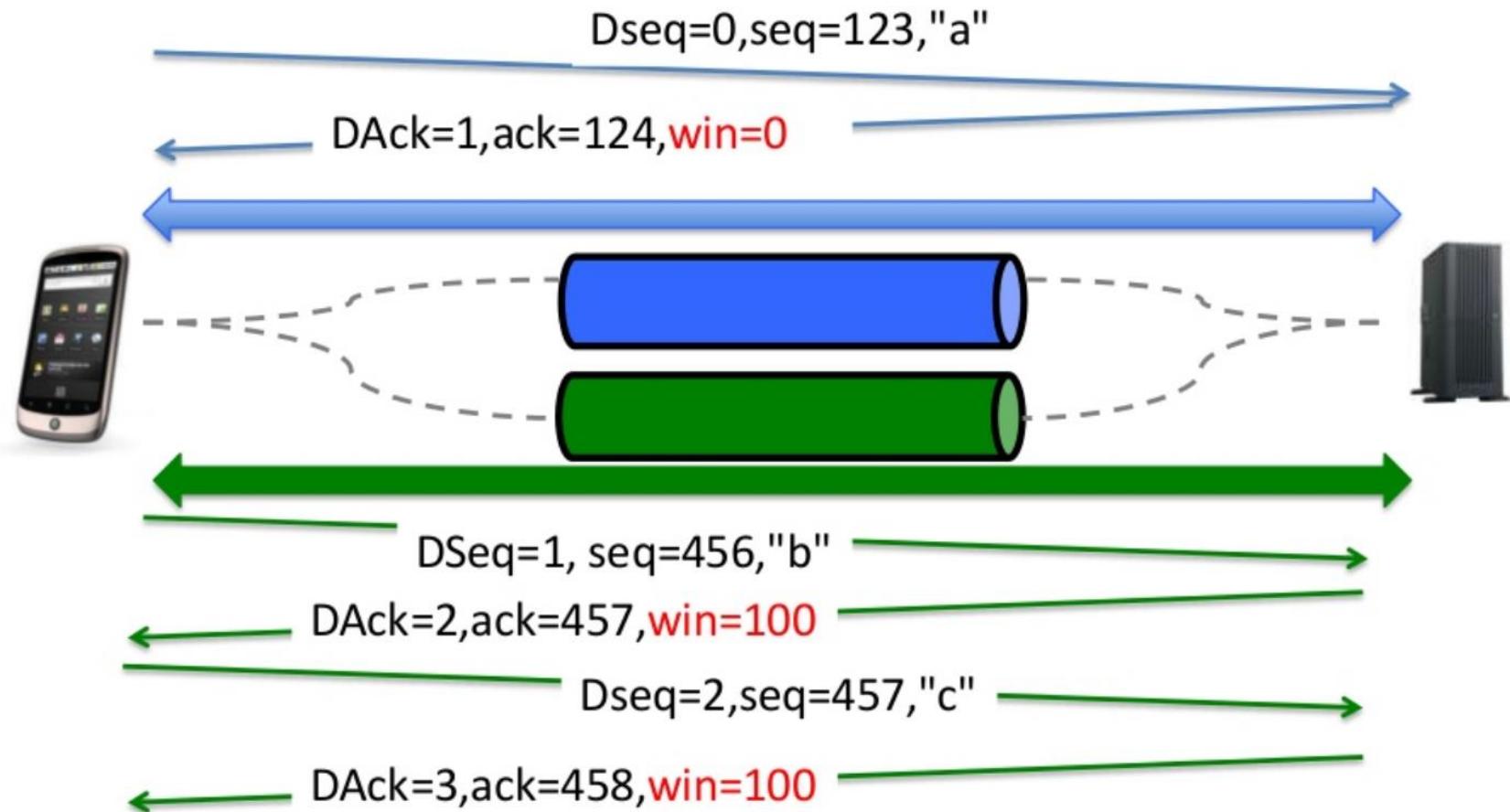
# MPTCP – Řízení toku ③

- Selhání subflow

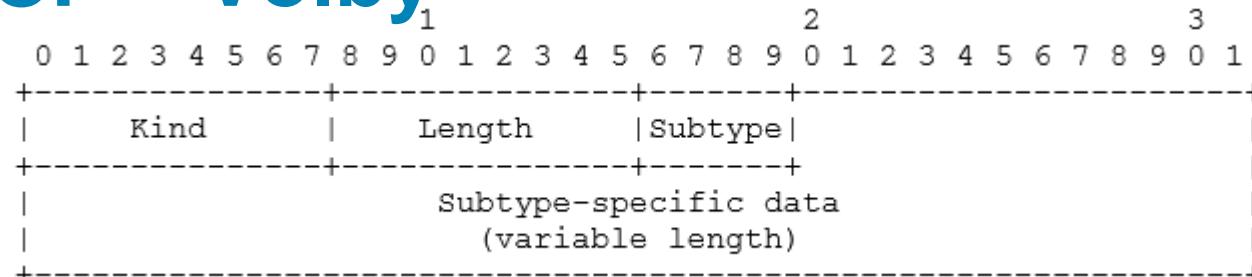


# MPTCP – Řízení záhlcení

- Nezávislá sliding window

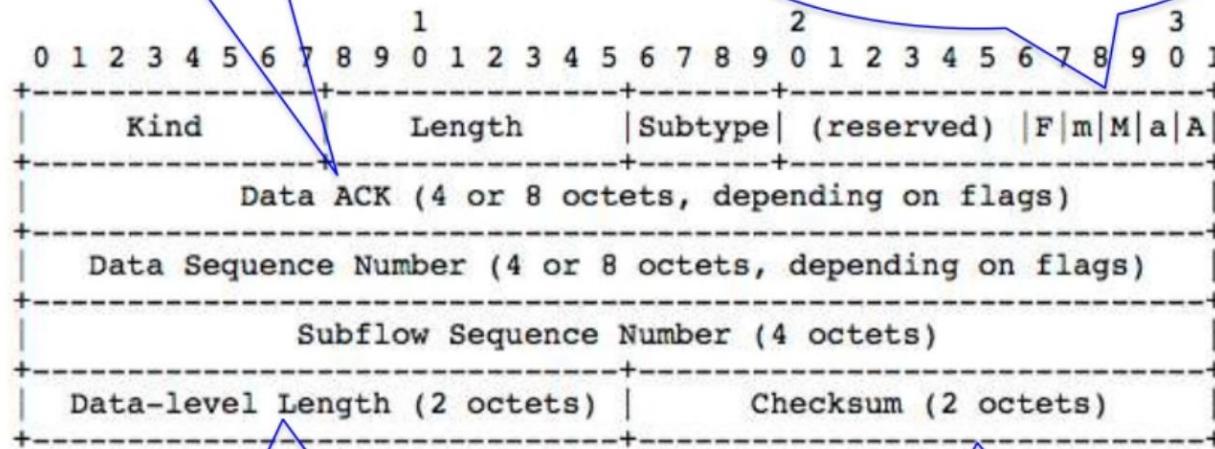


# MPTCP – Volby



Cumulative Data ack

A = Data ACK present  
a = Data ACK is 8 octets  
M = mapping present  
m = DSN is 8



Length of mapping, can extend beyond this segment

Computed over data covered by entire mapping + pseudo header

# MPTCP – Řízení zahlcení

- Subflow mají svázané řízení zahlcení
  - Udržují si ale separátní subflow congestion windows
- Additive Increase – Multiplicative Decrease
- MPTCP je férové s TCP
  - stejný BW na bottlenecku jako TCP nezávisle na počtu cest
    - Increase  $w_r$  for each ACK on path  $r$ , by

$$\min_{S \subseteq R : r \in S} \frac{\max_{s \in S} w_s / \text{RTT}_s^2}{\left( \sum_{s \in S} w_s / \text{RTT}_s \right)^2}$$

- Decrease  $w_r$  for each drop on path  $r$ , by  $w_r/2$

# MPTCP ve Wireshark'u

Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8) [siri-cap-long.pcap]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: tcp.stream eq 66

No.	Time	Source	Destination	Protocol	Length	Info
339981	467.769834000	1	4	TCP	90	49209 > https [SYN] Seq=0 Win=65535 Len=0
340012	467.801004000	1	2	TCP	78	https > 49209 [SYN, ACK] Seq=0 Ack=1 Win=
340013	467.802317000		4	TCP	74	49209 > https [ACK] Seq=1 Ack=1 Win=66240
340014	467.810746000		4	TLSv1	214	Client Hello

Frame 340013: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0

Ethernet II, Src: Apple\_16:88:ea (4c:b1:99:16:88:ea), Dst: Netgear\_d8:aa:90 (a0:21:b7:d8:aa:90)

Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 192.168.0.1 (192.168.0.1)

Transmission Control Protocol, Src Port: 49209 (49209), Dst Port: https (443), Seq: 1, Ack: 1, Len: 0

Source port: 49209 (49209)  
Destination port: https (443)  
[Stream index: 66]  
Sequence number: 1 (relative sequence number)  
Acknowledgment number: 1 (relative ack number)  
Header length: 40 bytes  
Flags: 0x010 (ACK)  
Window size value: 8280  
[Calculated window size: 66240]  
[Window size scaling factor: 8]  
Checksum: 0x7dd1 [validation disabled]  
Options: (20 bytes), Multipath TCP  
  Multipath TCP: Multipath Capable  
    Kind: Multipath TCP (30)  
    Length: 20  
    0000 .... = Multipath TCP subtype: Multipath Capable (0)  
    .... 0000 = Multipath TCP version: 0  
    Multipath TCP flags: 0x01  
      0.... .... = Checksum required: 0  
      .... ....1 = Use HMAC-SHA1: 1  
    Multipath TCP Sender's Key: 7424751654081284655  
    Multipath TCP Receiver's Key: 17925294879282179448  
  [SEQ/ACK analysis]

0000 a0 21 b7 d8 aa 90 4c b1 99 16 88 ea 08 00 45 00 .!....L. ....E.  
0010 00 3c 97 2f 40 00 40 06 c1 5c c0 a8 00 02 11 82 .<./@. .\.....  
0020 10 04 c0 39 01 bb 96 28 22 05 f8 ba fd 3a a0 10 ...9...("....:  
0030 20 58 7d d1 00 00 1e 14 00 01 67 0a 04 4a 97 02 X}..... .g..J..  
0040 4a 2f f8 c3 70 b8 6d be 2d 78 J/.p.m. -x..

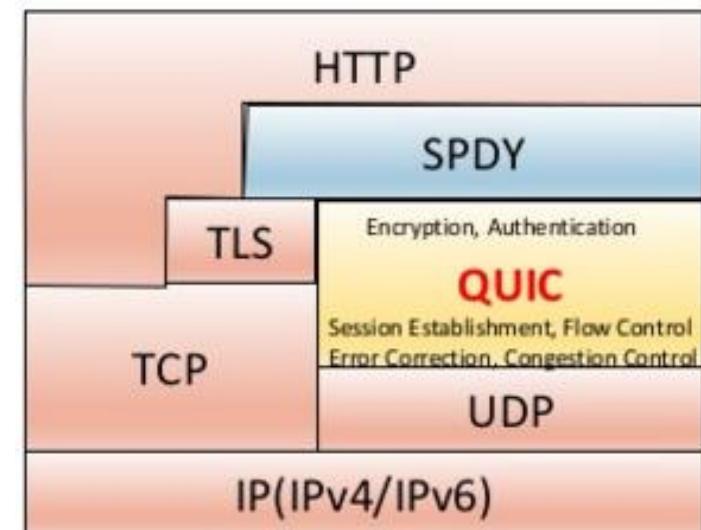
File: "/Users/obo/tmp/siri-cap.pcap" | Packets: 360139 | Displayed: 270 Ma... | Profile: Default

# MPTCP – Shrnutí

- *To samé, co umí TCP + vlastní řízení zahlcení*
- Congestion control
  - The Linked Increase Algorithm
  - The Opportunistic Linked Increase Algorithm
  - The wVegas delay based congestion control algorithm
  - The Balanced Linked Increase Algorithm

# Quick UDP Internet Connections

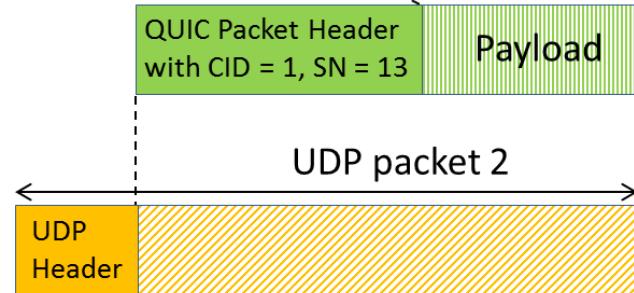
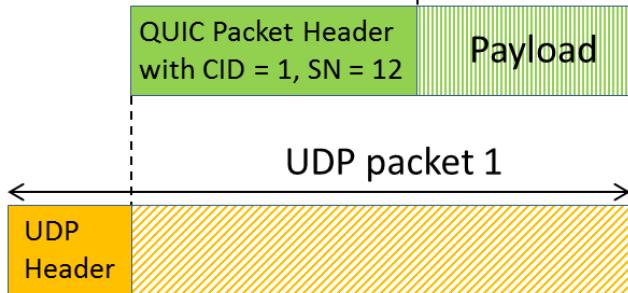
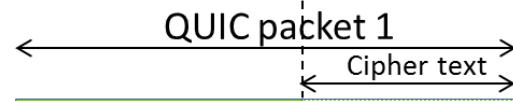
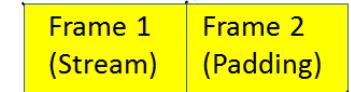
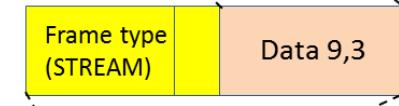
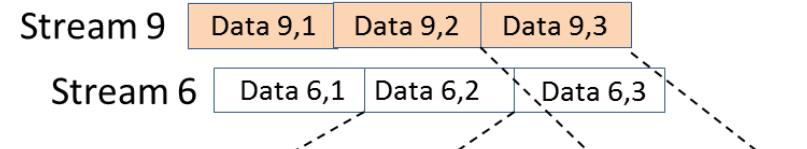
- Problematické navazování spojení u TCP
  - Útoky jako SYN Flood
  - $[1\frac{1}{2}|3] \times$  RTT před započetím komunikace
- <http://www.chromium.org/quic>
- Multiplexing spojení přes UDP
  - do několika streamů
- Lepší než SPDY
  - <http://dev.chromium.org/spdy/spdy-whitepaper>
  - Zrychlení načítání webu (komprese)
- Packet-pacing predikce bandwidth a proaktivní znova zasílání



# QUIC – Zpráva ①

Two QUIC Data Packets  
sent by the browser, each containing  
a different byte stream

Connection ID = 1  
Sequence Numbers: 12, 13  
Streams: 6 and 9



# QUIC – Zpráva ②

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1														
public flag (1)																																													
		connection id (0,1,4,8)																																											
		version(0,1)																																											
sequence num(1,4,6,8)		private flag(1)								FEC offset(0,1)																																			
frame_type(0x80~)		stream_id(1,2,3,4)																																											
offset(0, 2,3,4,5,6,7,8)															data length(0, 2)																														
Stream DATA																																													

# QUIC – Zpráva ③

## ■ Public flags

NONE	-	-	-	-	-	-	-	0
VERSION	-	-	-	-	-	-	-	1
RST	-	-	-	-	-	-	1	-
0BYTE_CONNECTION_ID	-	-	-	-	0	0	-	-
1BYTE_CONNECTION_ID	-	-	-	-	0	1	-	-
4BYTE_CONNECTION_ID	-	-	-	-	1	0	-	-
8BYTE_CONNECTION_ID	-	-	-	-	1	1	-	-
1BYTE_SEQUENCE	-	-	0	0	-	-	-	-
2BYTE_SEQUENCE	-	-	0	1	-	-	-	-
4BYTE_SEQUENCE	-	-	1	0	-	-	-	-
6BYTE_SEQUENCE	-	-	1	1	-	-	-	-

Version field is **no longer needed** after negotiation.

Use **8byte** Connection ID by default unless negotiated.

Sequence Number is fixed 64bit. but it can be sent **only 1byte** on the wire.

## ■ Private flags

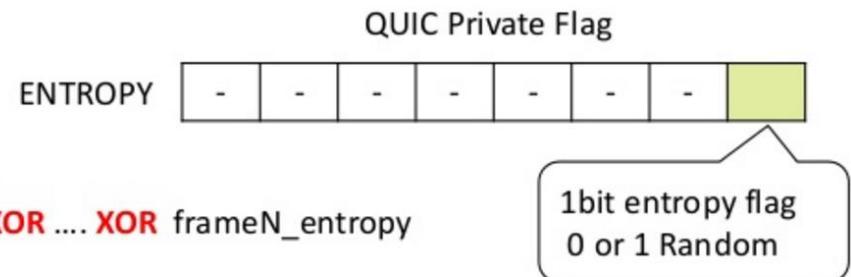
NONE	-	-	-	-	-	-	-	0
ENTROPY	-	-	-	-	-	-	-	1
FEC_GROUP	-	-	-	-	-	-	1	-
FEC	-	-	-	-	-	1	-	-

Random bit

this frame is in FEC group

FEC parity data is included

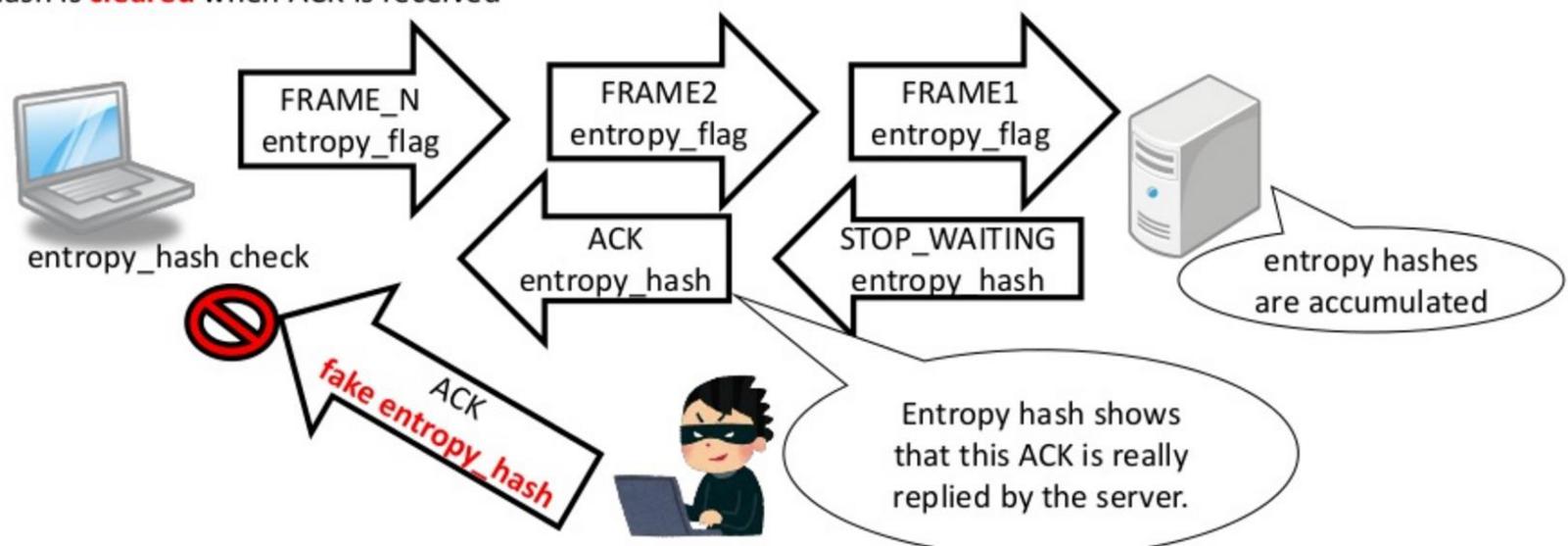
# QUIC – Entropy hash



1byte entropy: entropy flag << seq number%8

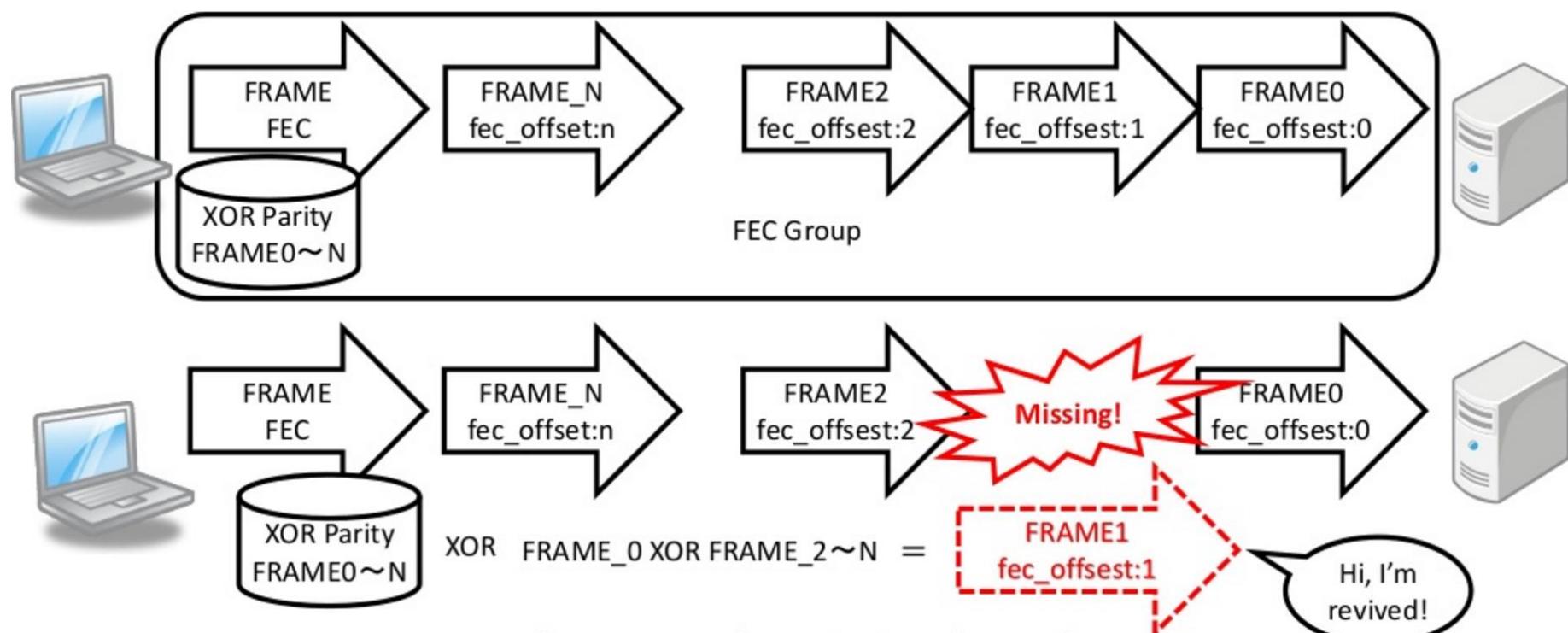
1byte entropy hash = frame1\_entropy **XOR** frame2\_entropy **XOR** .... **XOR** frameN\_entropy

entropy hash is **cleared** when ACK is received



# QUIC – FEC

- Jako u RAID disků, bez nutnosti znovuzasílat chybějící data!



# QUIC – Řízení toku ①

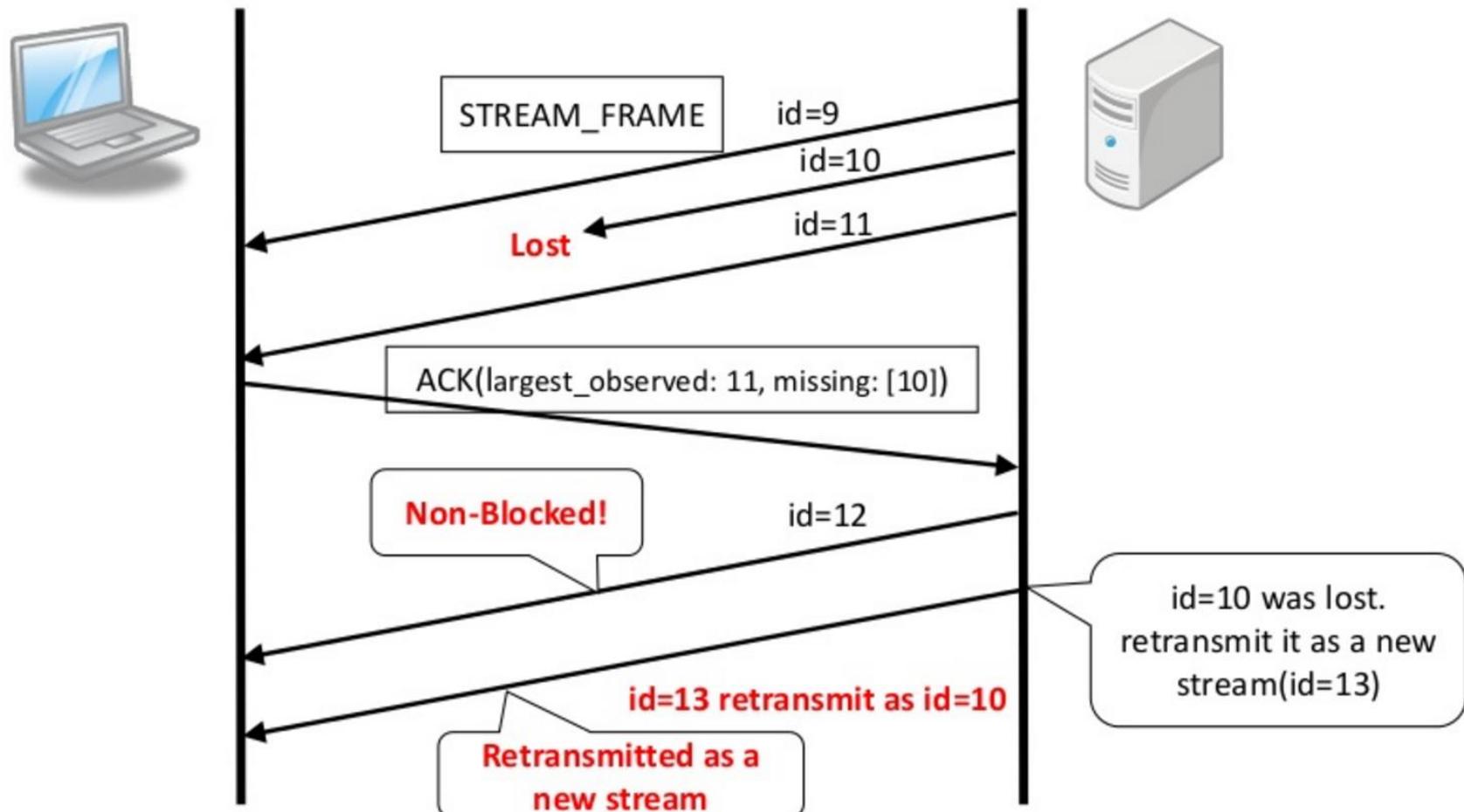
## ■ Potvrzení

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1										
public flag (1)																															
connection id (0,1,4,8)																															
		version(0,1)																													
		sequence num(1,4,6,8)					private flag(1)			frame_type(0x40~0x7f)																					
entropy hash(1)		Largest Observed					Largest Observed Delta Time(2)																								
num of received packet(1)		delta largest observed#1					time delta#1																								
							delta largest observed#2					time delta#2																			
		delta largest observed#3					time delta#3																								
num of missing packet(1)		missing packet seq#1					range length#1					missing packet seq#2																			
range length#2		missing packet seq#3					range length#3					num of revived packets(1)																			
revived packet seq#1		revived packet seq#2					revived packet seq#3					revived packet seq#4																			

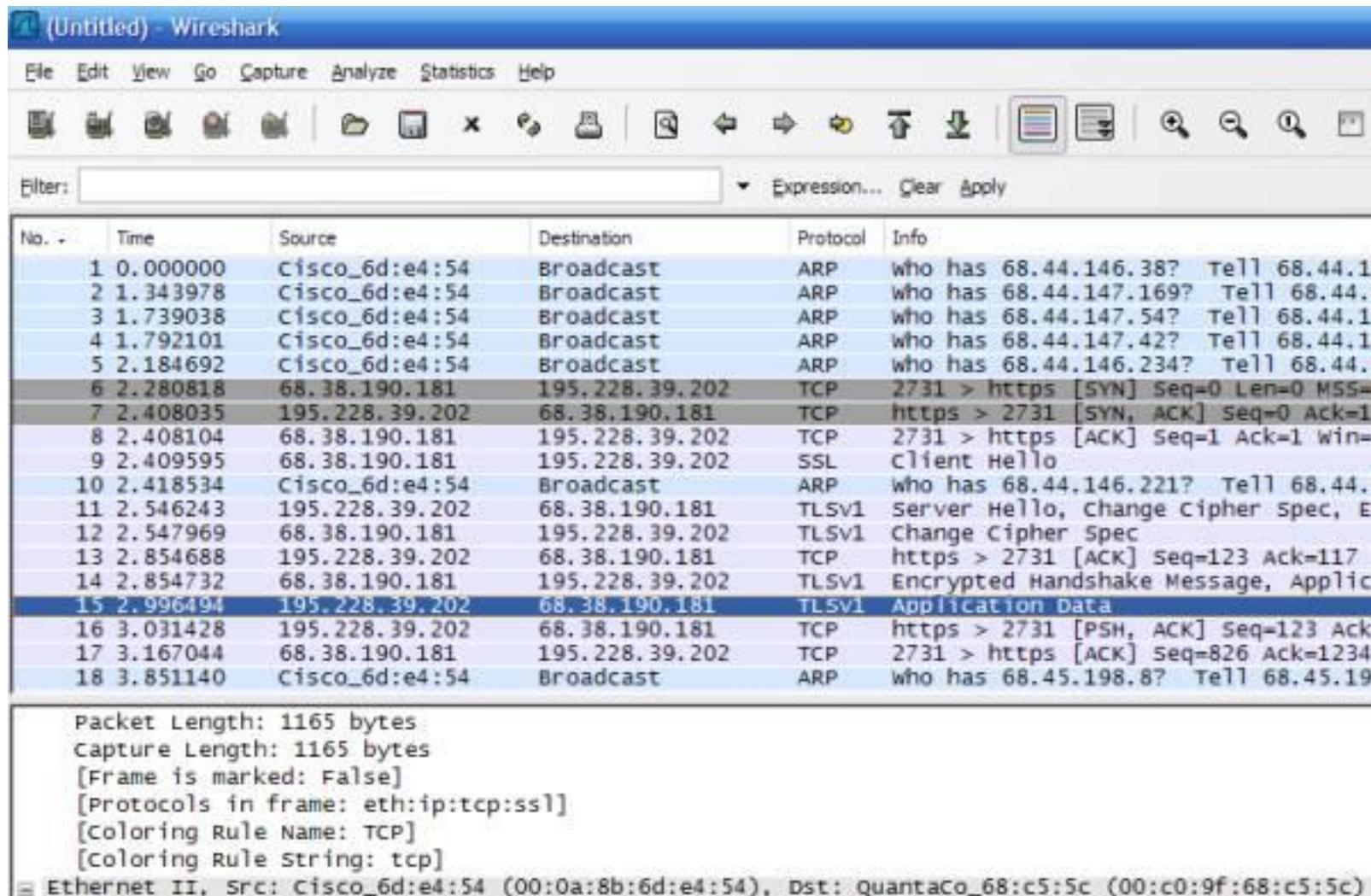
ACK conveys Largest Observed Seq, Packet Timestamp, Missing Packets and Revived Packets

# QUIC – Řízení toku ②

## ■ Znovuzaslání



# QUIC – TLS ve Wireshark



The screenshot shows a Wireshark capture window titled '(Untitled) - Wireshark'. The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Help. The toolbar contains icons for opening files, saving, zooming, and filtering. A 'Filter:' field is present, along with Expression..., Clear, and Apply buttons. The main pane displays a list of network packets. The columns are No., Time, Source, Destination, Protocol, and Info. The 'Info' column provides detailed protocol analysis for each packet. The captured traffic consists of 18 packets, starting with ARP broadcast frames and followed by a TCP handshake and a TLSv1 handshake.

No. +	Time	Source	Destination	Protocol	Info
1	0.000000	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.146.38? Tell 68.44.146.38
2	1.343978	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.147.169? Tell 68.44.147.169
3	1.739038	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.147.54? Tell 68.44.147.54
4	1.792101	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.147.42? Tell 68.44.147.42
5	2.184692	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.146.234? Tell 68.44.146.234
6	2.280818	68.38.190.181	195.228.39.202	TCP	2731 > https [SYN] Seq=0 Len=0 MSS=1460
7	2.408035	195.228.39.202	68.38.190.181	TCP	https > 2731 [SYN, ACK] Seq=0 Ack=1
8	2.408104	68.38.190.181	195.228.39.202	TCP	2731 > https [ACK] Seq=1 Ack=1 Win=64
9	2.409595	68.38.190.181	195.228.39.202	SSL	client Hello
10	2.418534	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.146.221? Tell 68.44.146.221
11	2.546243	195.228.39.202	68.38.190.181	TLSv1	Server Hello, Change Cipher Spec, ECDHE-RSA-AES128-GCM-SHA256, TLSv1.2, Compress None
12	2.547969	68.38.190.181	195.228.39.202	TLSv1	Change Cipher Spec
13	2.854688	195.228.39.202	68.38.190.181	TCP	https > 2731 [ACK] Seq=123 Ack=117 Win=64
14	2.854732	68.38.190.181	195.228.39.202	TLSv1	Encrypted Handshake Message, Application Data
15	2.996494	195.228.39.202	68.38.190.181	TLSv1	Application Data
16	3.031428	195.228.39.202	68.38.190.181	TCP	https > 2731 [PSH, ACK] Seq=123 Ack=124
17	3.167044	68.38.190.181	195.228.39.202	TCP	2731 > https [ACK] Seq=826 Ack=1234
18	3.851140	Cisco_6d:e4:54	Broadcast	ARP	who has 68.45.198.8? Tell 68.45.198.8

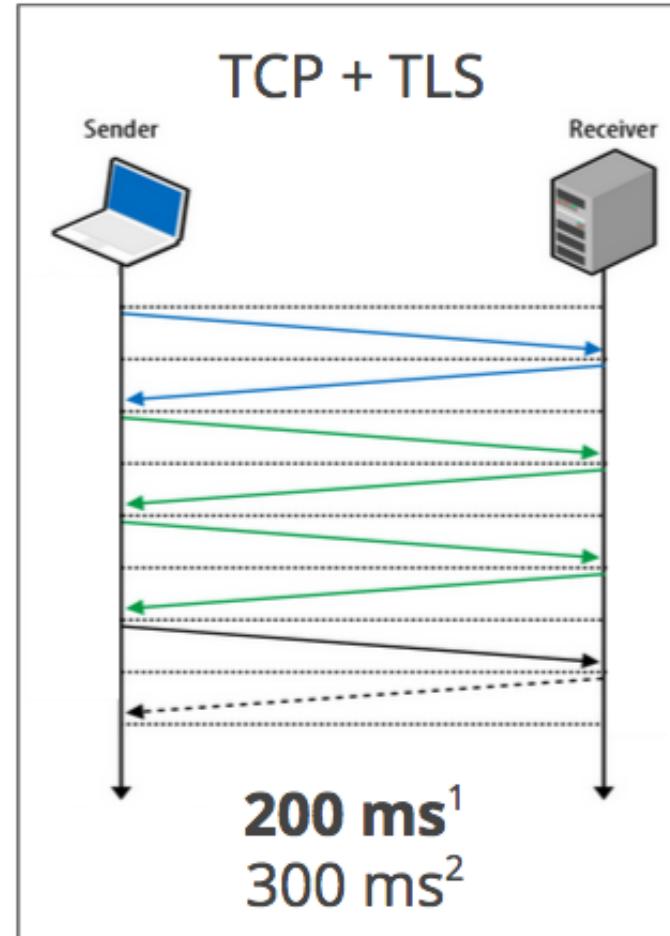
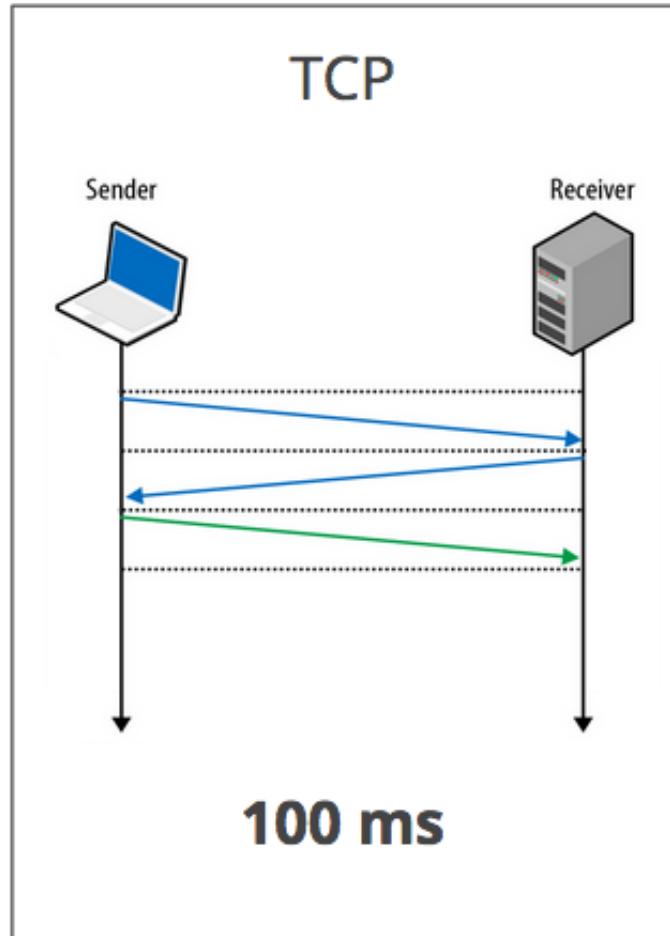
Packet Length: 1165 bytes  
Capture Length: 1165 bytes  
[Frame is marked: False]  
[Protocols in frame: eth:ip:tcp:ssl]  
[Coloring Rule Name: TCP]  
[Coloring Rule String: tcp]

Ethernet II, Src: Cisco\_6d:e4:54 (00:0a:8b:6d:e4:54), Dst: Quantaco\_68:c5:5c (00:c0:9f:68:c5:5c)

<https://www.owasp.org/images/b/bf/ReplayWireshark1.jpg>

# QUIC – RTT Porovnání

<https://jacobianengineering.com/blog/2016/10/1543/>



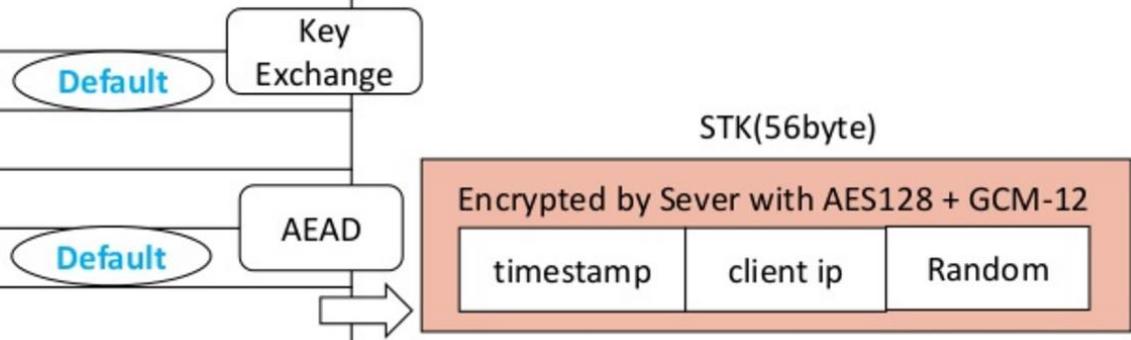
1. Repeat connection
2. Never talked to server before

# QUIC – Getting rid off SSL ①

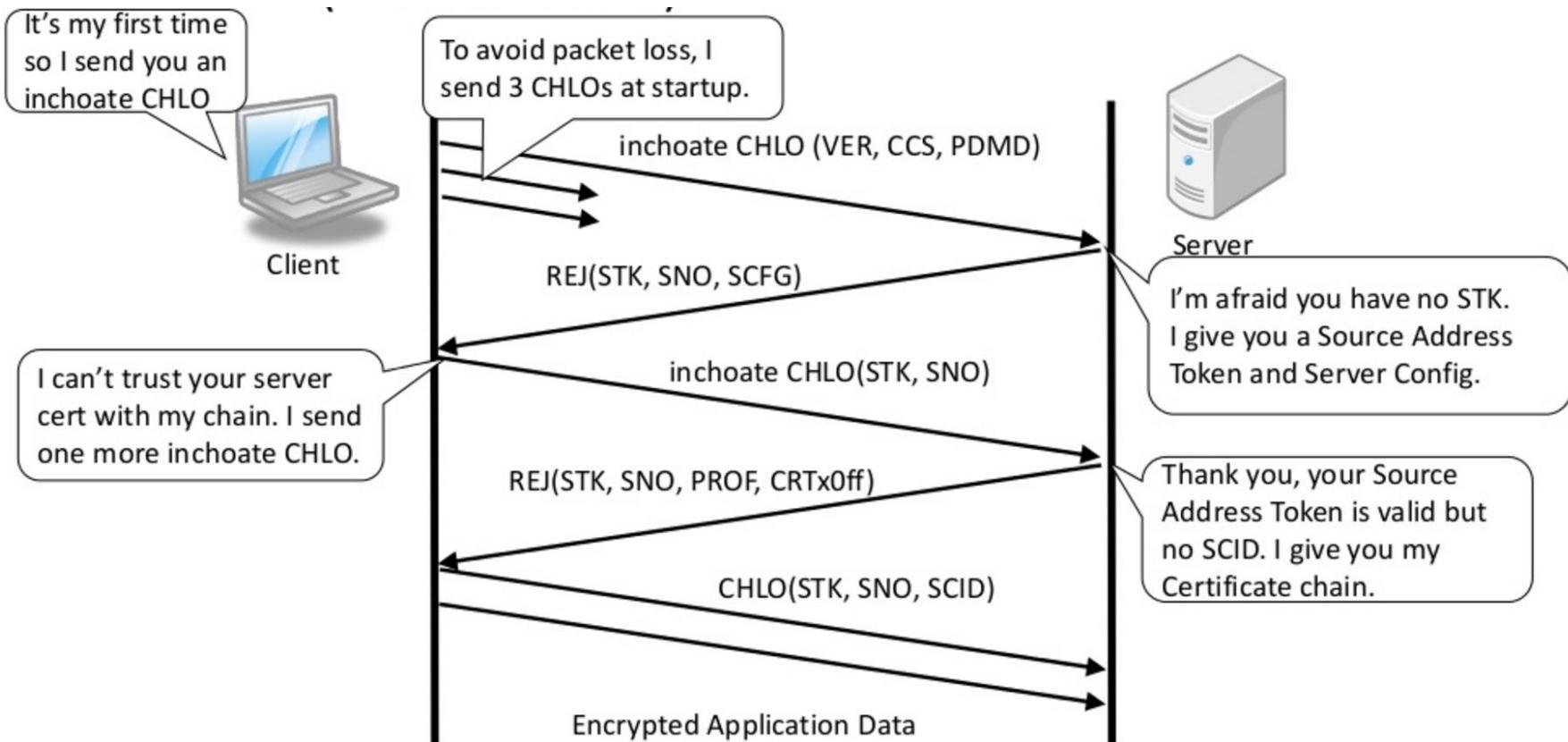
public flag (1)						
	connection id (0,1,4,8)					
	version(0,1)					
	sequence num(1,4,6,8)	private flag(1)	FEC offset(0,1)			
frame_type(0x80~)	stream_id = 0x01					
offset(0, 2~8)	data length(0, 2)					
FNV1a-128 hash truncated higher 32bit (12) snipped						
Message Tag(4)						
Number of Entries(2)	Padding(2)					
Tag#1(4)						
end_offset#1(4)						
Tag#2(4)						
end_offset#2(4)						
Value#1						
Value#2						

# QUIC – Getting rid off SSL ②

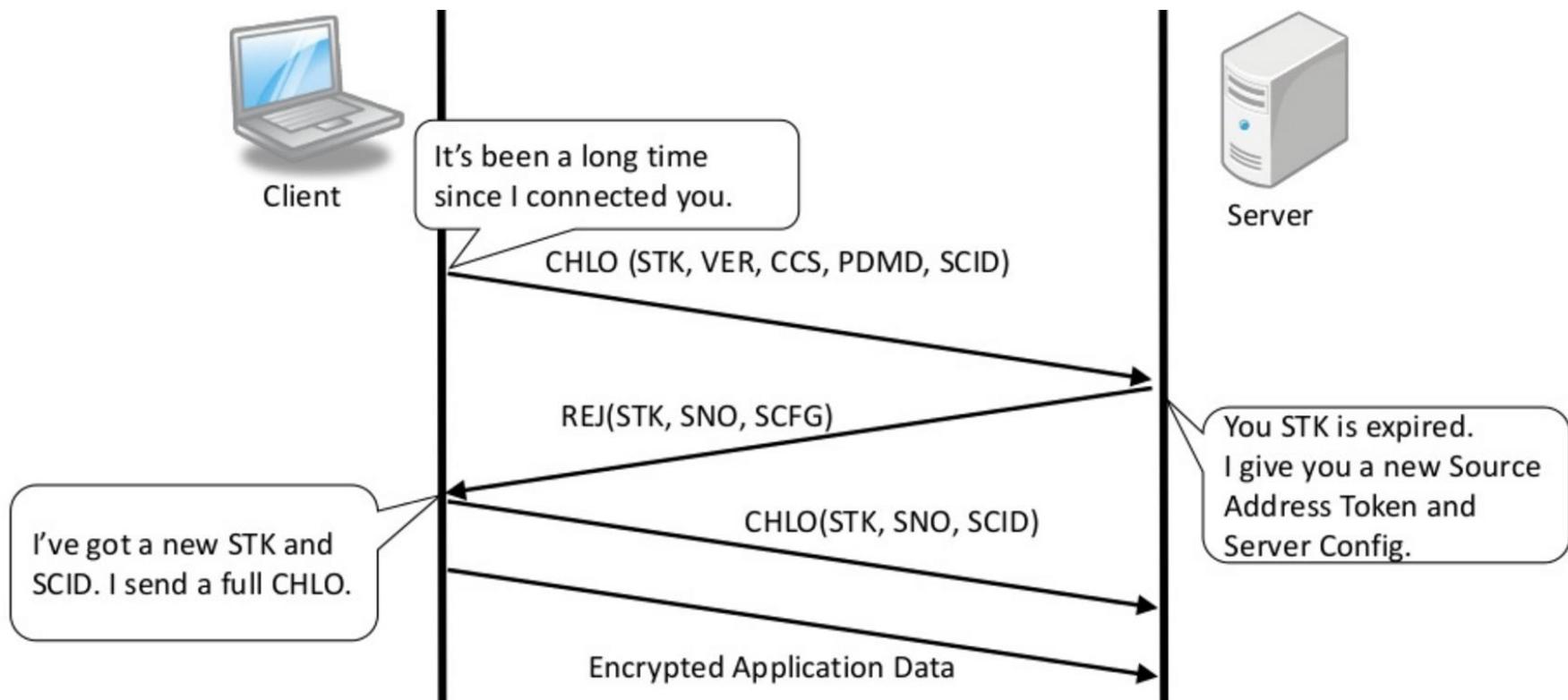
Crypto Tag	Meaning
<b>CHLO</b>	<b>Client Hello</b>
<b>SHLO</b>	<b>Server Hello</b>
SCFG	Server Config
<b>REJ</b>	<b>Reject</b>
CETV	Client encrypted tag-value pairs
PRST	Public Reset
<b>SCUP</b>	<b>Server Config Update</b>
P256	ECDH Curve P-256
C255	ECDH Curve25519
NULL	null algorithm
AESG	AES128 + GCM-12
CC12	ChaCha20 + Poly1305
STK	Source Address Token



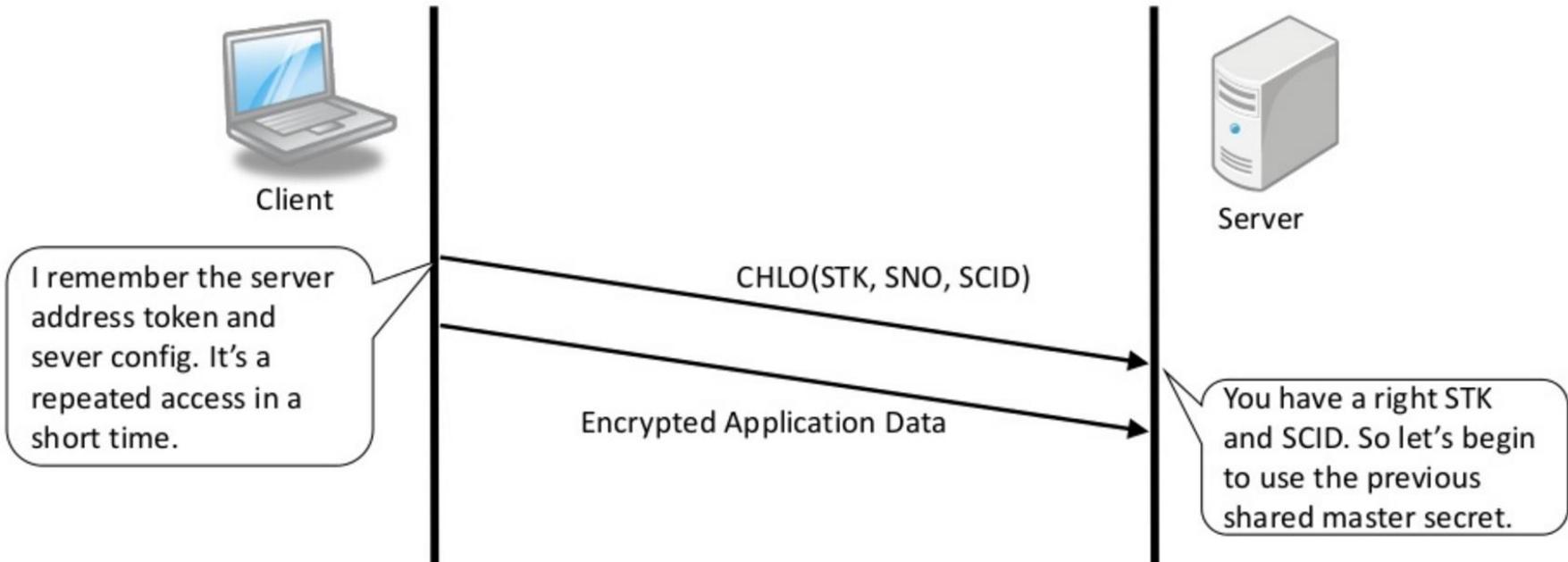
# QUIC – 2xRTT HTTPS



# QUIC – 1xRTT HTTPS

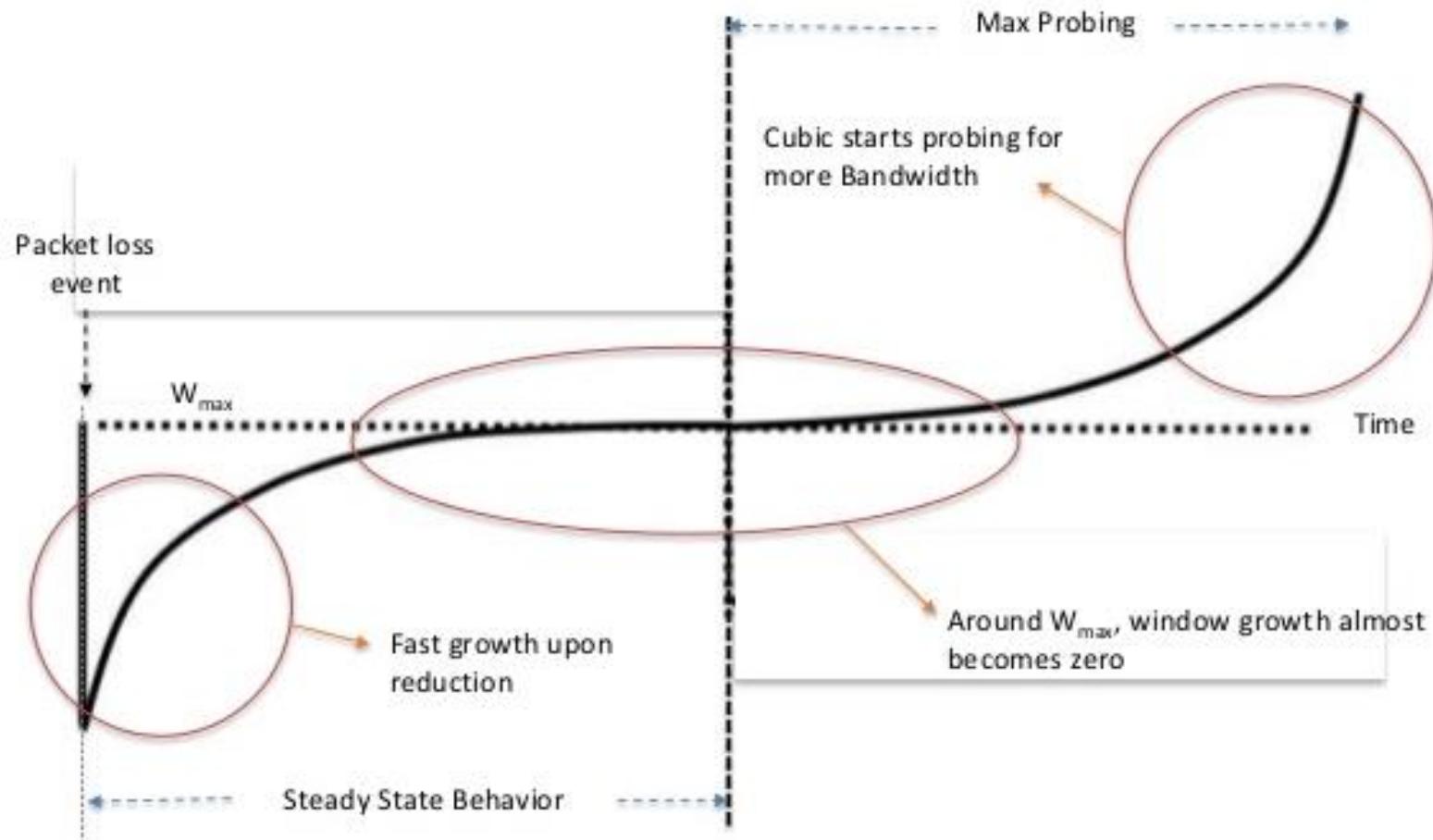


# QUIC – 0xRTT HTTPS



# QUIC – Řízení zahlcení

- TCP CUBIC ve výchozím stavu



<http://image.slidesharecdn.com/cubickdw-150301075845-conversion-gate02/95/cubic-17-638.jpg?cb=1425196889>

# QUIC ve Wireshark'u

The screenshot shows a Wireshark capture window titled "quic\_new2.pcapng". The packet list pane displays 11772 total packets, 11772 displayed, 0 dropped, and 0 load time. The selected packet is a Client Hello message (packet 1) from source 10.44.100.45 to destination 216.58.211.68. The packet details pane shows the following fields:

No.	Time	Source	Destination	Protocol	Length	Version	Tag	Info
1	0...	10.44.100.45	216.58.211.68	QUIC	1392	Q025		CHLO Client Hello, CID: 11414687164953879775, Seq: 1
2	0...	216.58.211.68	10.44.100.45	QUIC	1392		REJ	Rejection, CID: 11414687164953879775, Seq: 1
3	0...	216.58.211.68	10.44.100.45	QUIC	1392			Payload (Encrypted), CID: 11414687164953879775, Seq: 2
4	0...	216.58.211.68	10.44.100.45	QUIC	79			Payload (Encrypted), CID: 11414687164953879775, Seq: 3
5	0...	10.44.100.45	216.58.211.68	QUIC	82			Payload (Encrypted), CID: 11414687164953879775, Seq: 2
6	0...	10.44.100.45	216.58.211.68	QUIC	1392			CHLO Client Hello, CID: 11414687164953879775, Seq: 3
7	0...	216.58.211.68	10.44.100.45	QUIC	1392			Payload (Encrypted), Seq: 4
8	0...	10.44.100.45	216.58.211.68	QUIC	82			Payload (Encrypted), CID: 11414687164953879775, Seq: 4
9	0...	10.44.100.45	216.58.211.68	QUIC	480			Payload (Encrypted), CID: 11414687164953879775, Seq: 5
10	0...	216.58.211.68	10.44.100.45	QUIC	74			Payload (Encrypted), Seq: 5

The packet details pane shows the Client Hello payload structure:

- Tag Number: 15
- Padding: 0000
- Tag/value: PAD (Padding) (l=1063)
- Tag/value: SNI (Server Name Indication) (l=14): www.google.com
- Tag/value: VER (Version) (l=4): Q025
- Tag/value: CCS (Common Certificate Sets) (l=8)
- Tag/value: MSPC (Max streams per connection) (l=4): 100
  - Tag Type: MSPC (Max streams per connection)
  - Tag offset end: 1093
  - [Tag length: 4]
  - Tag/value: 64000000
  - Max streams per connection: 100
- Tag/value: UAID (Client's User Agent ID) (l=47): canary Chrome/47.0.2517.0 Windows NT 6.2; WO64
- Tag/value: TCID (Connection ID truncation) (l=4)
  - Tag value: 0000\_0000\_0000\_0000

The bytes pane shows the raw hex and ASCII data for the selected Client Hello packet (packet 1). The bytes pane also highlights the TCID truncation field at offset 0x0000.

# QUIC – Shrnutí

- Reliable delivery
  - Inorder byte stream
- Flow control
  - SMART s kumulativním potvrzováním
- Congestion control
  - TCP CUBIC
- Error correction
  - FEC
- V nejlepším případě connection-less, jinak connection-oriented

# Watson's Delta-t

- <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=65288>
- „Všechna spojení existují stále“
  - Kontext je ale třeba udržovat jen k těm aktivním
  - Po určité době se kontext zresetuje
- Bezpečný přenos bez počáteční synchronizace
  - SYN, FIN jsou zbytečné, všechny protokoly jsou soft-state
- Portové číslo je 64bitové
- Omezení tří časovačů
  - Pokud zařízení všechna zařízení v síti mají stejné hodnoty těchto timerů, je samosynchronizující se

# $\Delta t$ - Basics

- Data assurance through data sequence numbers (SN)
- Left Window Edge – (R)LWE
- $SN < LWE$  = duplicate
- 3 message types
  - Data
  - Control (ack)
  - Direct control (rendezvous)

# $\Delta t$ - Timers

- MPL – Maximum Packet Lifetime
- R – Maximum time to attempt retransmission
- A – Maximum time a receiver will wait before sending an Ack
- n – number of bits for sequence number
- T – maximum transmission rate
- Assurance that a sender generating SN at the maximum speed will not reuse an SN until it is guaranteed that an SN and any acks no longer exists in the network
  - $2^n > (2 * \text{MPL} + R + A) * T$
- $\Delta t = \text{MPL} + R + A$
- RTimer – receiver part connection record
- STimer – sender part connection record

# Connection management phases

1. Initializing (opening) connection ends to non-default
2. Evolving the state during data transfer
3. Resetting or terminating (closing) state information
  - Initial SN must meet opening conditions:
    - 1.O1 if no connection exists, no packets from previous connection should cause it to be initialized and duplicate data to be accepted
    - 2.O2 If connection exists, no packet from previous connection should be acceptable within current connection
  - Connection should be closed via graceful shutdown (avoiding ambiguity)
    - 1.C1 A receiving side must not close until it has received all of sender's possible retransmissions and can unambiguously respond to them
    - 2.C2 A sending side must not close until it has rcvd an Ack of its final retransmission to return before reporting a giveup failure.

# Timer rulez!

- $S\text{Timer} = 3 * \Delta t$ 
    - Refreshed whenever new data SN is sent or reliable Ack
  - $R\text{timer} = 2 * \Delta t$ 
    - Refreshed whenever a new SN is accepted or data overflow occur
- 
- R1: STimer is refreshed whenever a new SN or reliable Ack is sent
  - R2: When bit  $b_i$  exhausts ReTX count, no new data bit can be send (new SN)
  - R3: RTimer is refreshed whenever new SN is accepted or data overflow occurs
  - R4: When RTimer expires, the receive state is reset to its default values
  - R5: Once a bit or reliableAck is initially transmitted its lifetime is set equal to  $\Delta t$  and begins counting down
  - R6: Once a bit is tested for acceptance, the lifetime for generated Ack packet is set to  $\Delta t$
  - R7: When STimer expires, the state is reset and if  $\text{ReTX} != \text{empty}$  giveup error is reported. *Always reports error and restarts STimer.*

# RTimer conditions

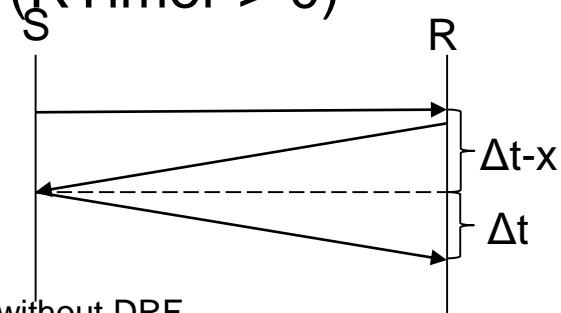
1.(Assurance) No duplicates can be accepted.

- $\text{RTimer} > \Delta t$ ; no bit can live longer than  $\Delta t$  by R5

2.(Smooth flow) Guarantee that any packet sent without DRF sent after packet with DRF will be accepted ( $\text{RTimer} > 0$ )

- $\text{RTimer} > 2\Delta t$

1. Packet with DRF set is sent from S to R in instant time
2. Ack arrived at  $\Delta t-x$
3.  $\Delta t-x$  is the last moment when new SN can be emitted due to rule R2 without DRF
4. Worst case it arrives at  $2\Delta t-x$ . RTimer must be  $> 0$  for any packet without DRF to be accepted



# STimer conditions

- Assurance

1. Allow graceful close – do not close until all data or packets sent needing Acks can be acknowledged

2. Assure that no SN will be reused until all previous packets or their Acks using the SN have died

- STimer >  $2\Delta t$ ;

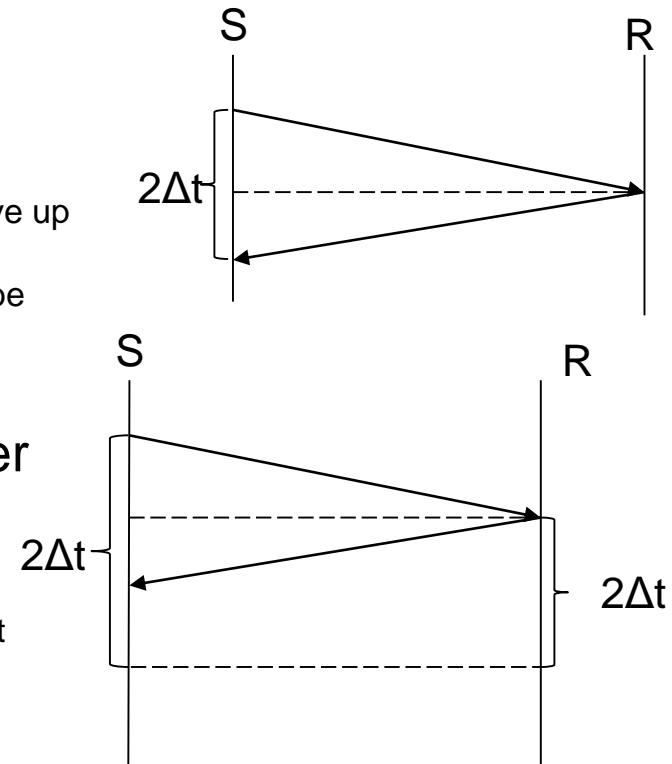
1. Packet can live at most  $\Delta t$  by rule R5. A data and its ack can live up to max  $2\Delta t$

2. For the same reason, if Ack is ever going to be received it will be within  $2\Delta t$

- Smooth flow Run equal to or longer than RTimer to guarantee acceptable SN's are generated

- Stimer >  $3\Delta t$

1. At most  $\Delta t$  to reach receiver which starts RTimer for another  $2\Delta t$



# Self-Check

# Otázky

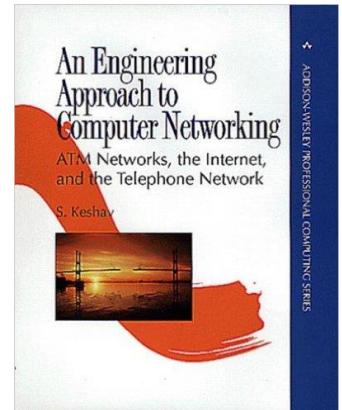
- Jmenuj a vysvětli aspoň tři chyby paketů!
- Jaké znáš techniky znova zasílání paketu?
- Uvažujme prostor sekvenčních čísel, který používá IP, maximální rychlostí linky 2 Mbps,  $T = A = 500$  ms. Jaké je v tomto scénáři největší povolené  $MPL$  pro pakety velikosti 40 B?
- Odesilatel zaslal pakety se sekvenčními čísly 10 až 15, příjemce odpověděl NACK pro paket s číslem 12. Jak se zachová odesilatel, pokud bude používat algoritmus Go-Back-N a jak v případě selektivního znova zaslání?
- Jak se určuje hodnota retransmission timeoutu?
- Co je to řízení spojení, řízení toku a řízení zahlcení?
- Co je to slow start, congestion avoidance a fast recovery?
- Jaké služby transportní vrstvy nabízí UDP, TCP, MPTCP, SCTP, DCCP, QUIC?

# Bibliografie

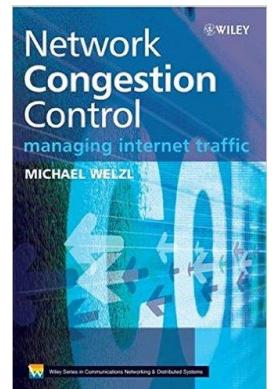
# Co si přečíst?

- Srinivasan Keshav

- <http://blizzard.cs.uwaterloo.ca/keshav/wiki/index.php/Books>
- [http://www.cs.cornell.edu/skeshav/book/slides/error\\_control/error\\_control.pdf](http://www.cs.cornell.edu/skeshav/book/slides/error_control/error_control.pdf)
- [http://www.cs.cornell.edu/skeshav/book/slides/flow\\_control/flow\\_control.pdf](http://www.cs.cornell.edu/skeshav/book/slides/flow_control/flow_control.pdf)



- Michael Welzl, Network Congestion Control: Managing Internet Traffic 1st Edition, ISBN-10: 047002528X



- <https://tools.ietf.org/html/rfc2988>
- <https://tools.ietf.org/html/rfc6298>
- <http://nes.fit.vutbr.cz/ivesely/pds/>

# Kam dál?

- [http://media.pearsoncmg.com/aw/aw\\_kurose\\_network\\_2/applets/go-back-n/go-back-n.html](http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/go-back-n/go-back-n.html)
- [http://media.pearsoncmg.com/aw/aw\\_kurose\\_network\\_4/applets/SR/index.html](http://media.pearsoncmg.com/aw/aw_kurose_network_4/applets/SR/index.html)
- <http://www.networkworld.com/news/tech/2005/112805techupdate.html>
- [http://en.wikipedia.org/wiki/Go-Back-N\\_ARQ](http://en.wikipedia.org/wiki/Go-Back-N_ARQ)
- [http://en.wikipedia.org/wiki/Selective\\_Repeat\\_ARQ](http://en.wikipedia.org/wiki/Selective_Repeat_ARQ)
- <http://www.slideshare.net/obonaventure/multipath-tcp>
- <http://www.slideshare.net/PeterREgli/overview-of-sctp-transport-protocol>
- <http://web.stanford.edu/class/cs340v/papers/dccp.pdf>
- [http://www.slideshare.net/shigeki\\_ohtsu/quic-overview](http://www.slideshare.net/shigeki_ohtsu/quic-overview)
- <http://slideplayer.com/slide/8611906/>