TCP / IP

Marcos Monteiro,
MBA, ITIL V3, Perito computacional Forense

http://www.marcosmonteiro.com.br
contato@marcosmonteiro.com.br

A

Internet

nasceu da

internet

História

História da comunicação da Internet

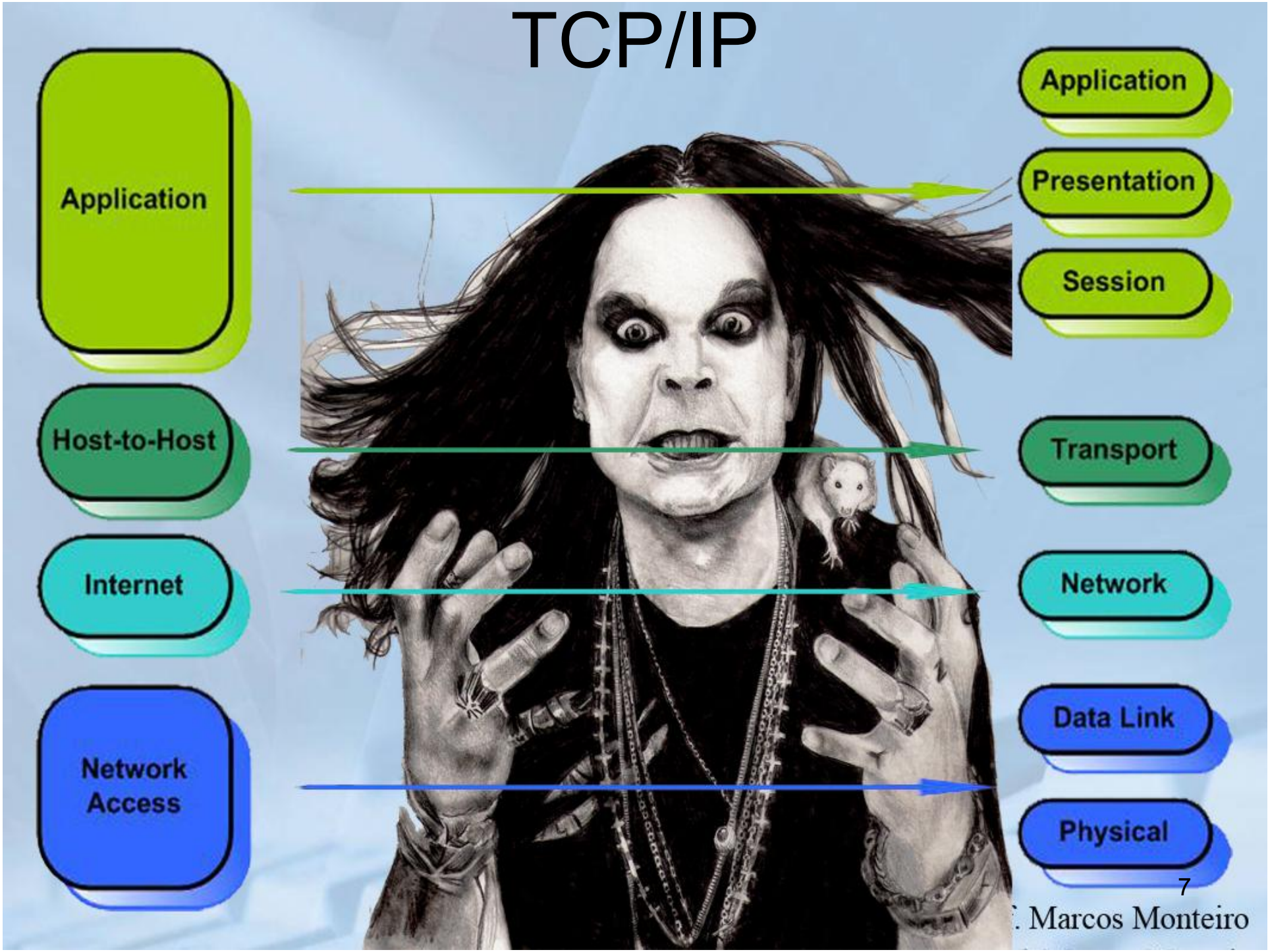
...e o TCP/IP ?

Se lembra do modelo OSI?

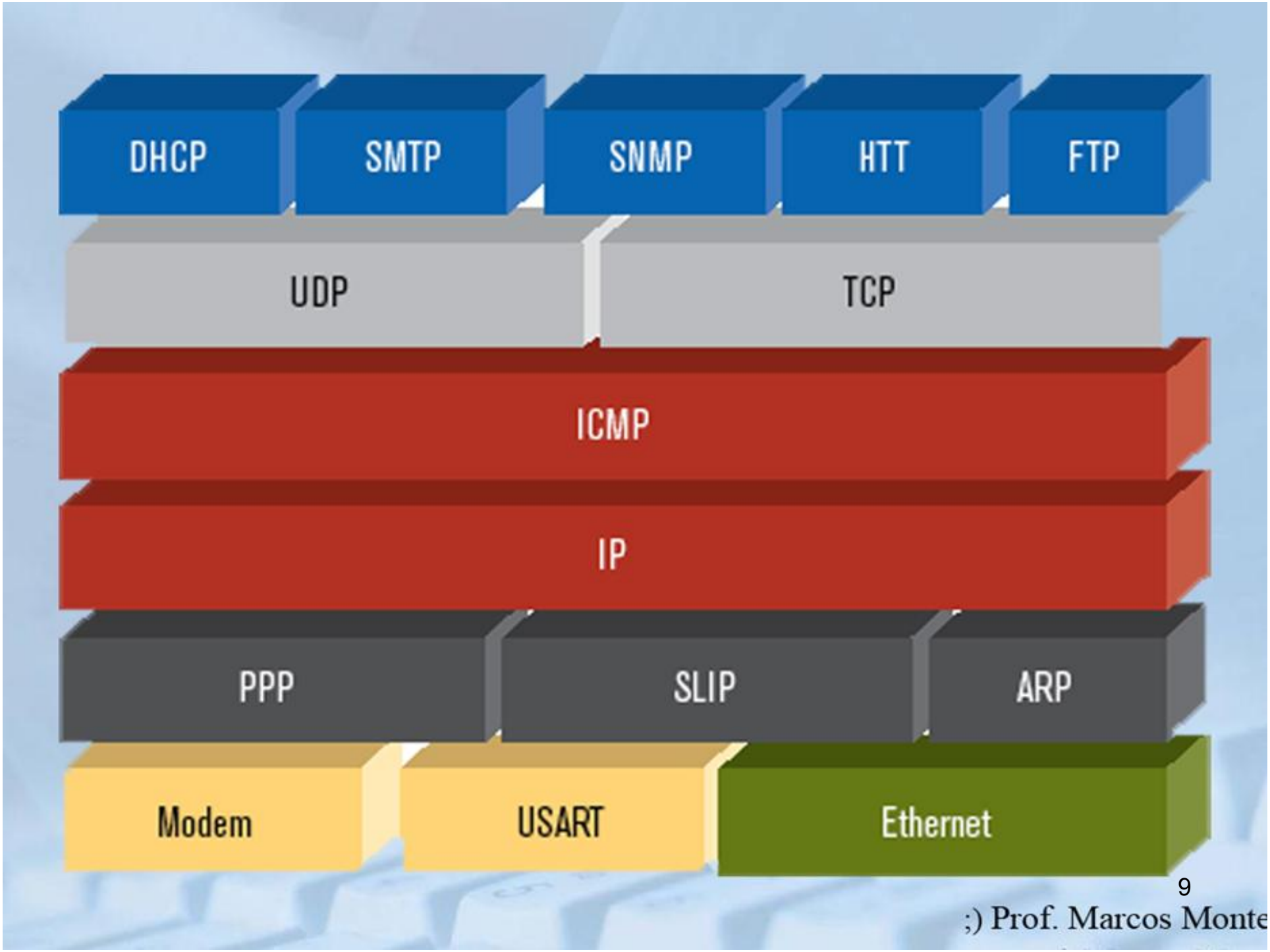
Se lembra do modelo OSI?

Camada	Exemplos
7 - Aplicação	HL7, Modbus
6 - Apresentação	TDI, ASCII, EBCDIC, MIDI, MPEG
5 - Sessão	Named Pipes, NetBIOS, SIP, SAP, SDP
4 - Transporte	NetBEUI
3 - Rede	NetBEUI, Q.931
2 - Ligação de dados	Ethernet, Token Ring, FDDI, PPP, HDLC, Q.921, Frame Relay, ATM, Fibre Channel
1 - Físico	RS-232, V.35, V.34, Q.911, T1, E1, 10BASE-T,100BASE-TX , ISDN, SONET, DSL

TCP/IP


Application
Host-to-Host
Internet
Network Access
Application
Presentation
Session
Transport
Network
Data Link
Physical

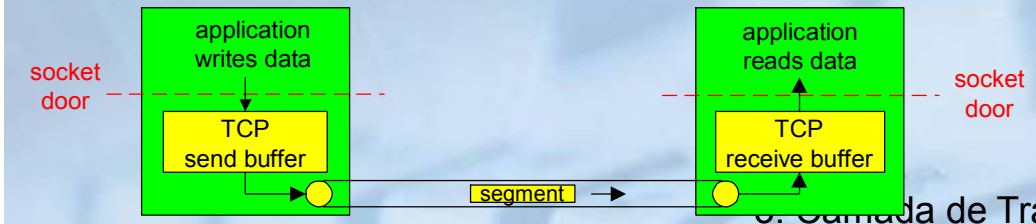
Camada	Exemplos	suite TCP/IP
7 - Aplicação	HL7, Modbus	HTTP, SMTP, SNMP, FTP, Telnet, NFS, NTP, BOOTP, DHCP, RMON, TFTP, POP3, IMAP, HTTP, TELNET
6 - Apresentação	TDI, ASCII, EBCDIC, MIDI, MPEG	XDR, SSL, TLS
5 - Sessão	Named Pipes, NetBIOS, SIP, SAP, SDP	Estabelecimento da sessão TCP
4 - Transporte	NetBEUI	TCP, UDP, RTP, SCTP
3 - Rede	NetBEUI, Q.931	IP, ICMP, IPsec, RIP, OSPF, BGP,
2 - Ligação de dados	Ethernet, Token Ring, FDDI, PPP, HDLC, Q.921, Frame Relay, ATM, Fibre Channel	MTP-2,ARP
1 - Físico	RS-232, V.35, V.34, Q.911, T1, E1, 10BASE-T,100BASE-TX , ISDN, SONET, DSL	


DHCP
SMTP
SNMP
HTT
FTP
UDP
TCP
ICMP
IP
PPP
SLIP
ARP
Modem
USART
Ethernet

Camada	Protocolo
5.Aplicação	HTTP, SMTP, FTP, SSH, RTP, Telnet, SIP, RDP, IRC, SNMP, NNTP, POP3, IMAP, BitTorrent, DNS, Ping ...
4.Transporte	TCP, UDP, SCTP, DCCP ...
3.Rede	IP (IPv4, IPv6) , ARP, RARP, ICMP, IPSec ...
2.Enlace	Ethernet, 802.11 WiFi, IEEE 802.1Q, 802.11g, HDLC, Token ring, FDDI, PPP,Switch ,Frame Relay,
1.Física	Modem, RDIS, RS-232, EIA-422, RS-449, Bluetooth, USB, ...

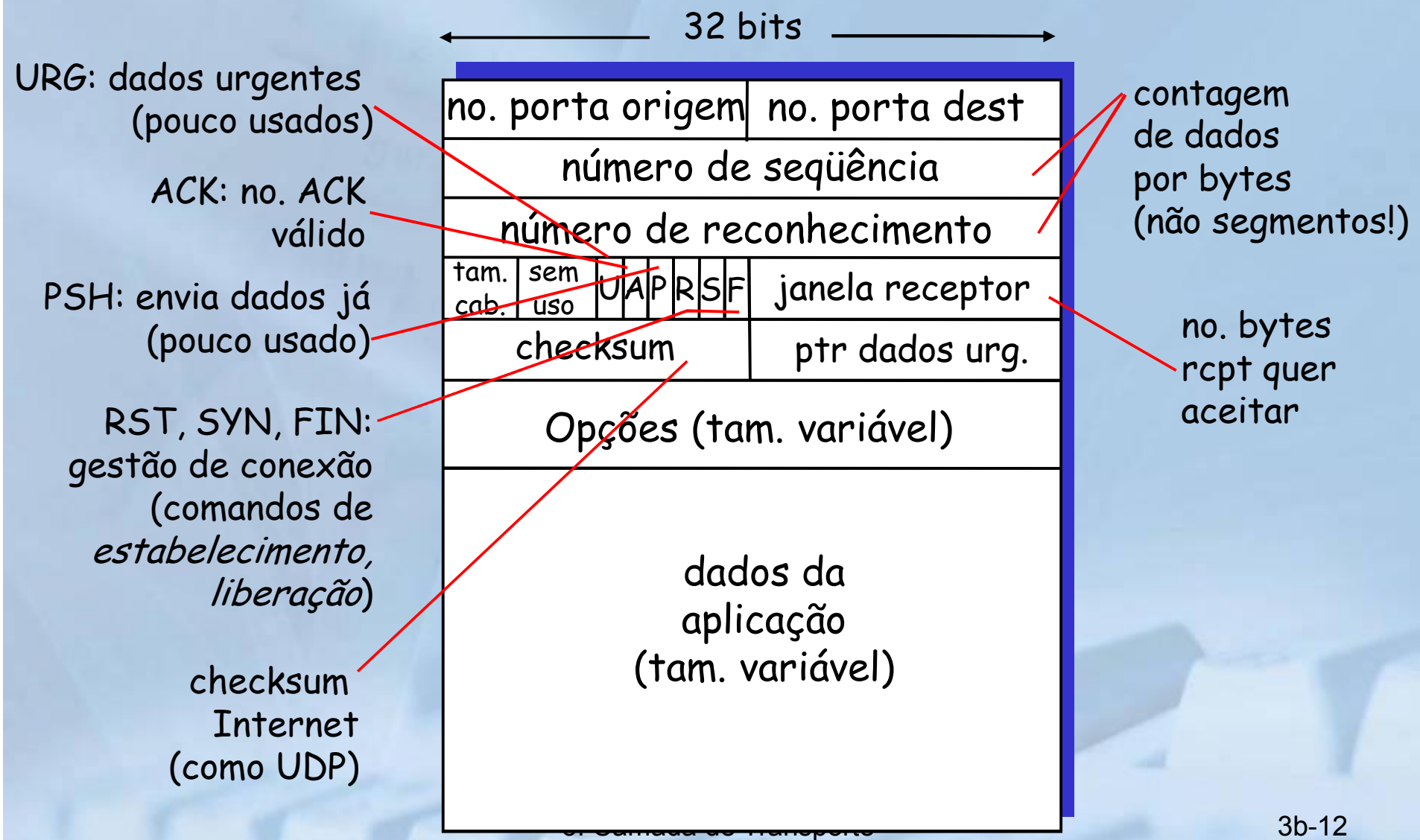
TCP: Visão geral RFCs: 793, 1122, 1323, 2018, 2581

- ponto a ponto:
  - 1 remetente, 1 receptor
- fluxo de bytes, ordenados, confiável:
  - não estruturado em msgs
- com paralelismo *(pipelined)*:
  - tam. da janela ajustado por controle de fluxo e congestionamento do TCP
- *buffers de envio e recepção*



- transmissão full duplex:
  - fluxo de dados bi-direcional na mesma conexão
  - MSS: tamanho máximo de segmento
- orientado a conexão:
  - handshaking (troca de msgs de controle) inicia estado de remetente, receptor antes de trocar dados
- fluxo controlado:
  - receptor não será afogado

TCP: estrutura do segmento



TCP: nos. de seq. e ACKs

Nos. de seq.:

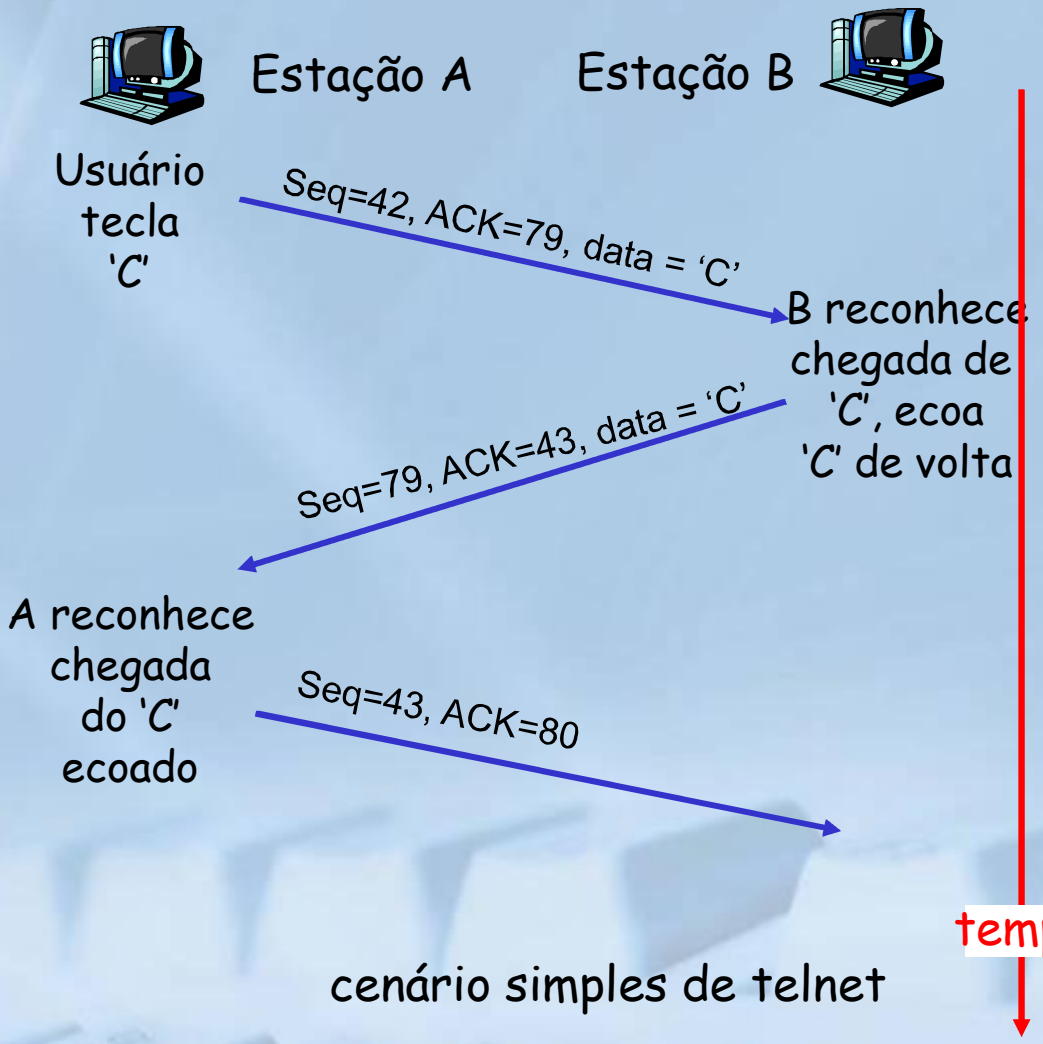
- "número"dentro do fluxo de bytes do primeiro byte de dados do segmento

ACKs:

- no. de seq do próx. byte esperado do outro lado
- ACK cumulativo

P: como receptor trata segmentos fora da ordem?

- R: espec do TCP omissa - deixado ao implementador



cenário simples de telnet

TCP: Tempo de Resposta (RTT – *Round Trip Time*) e Temporização

P: como escolher valor do temporizador TCP?

- maior que o RTT
  - note: RTT pode variar
- muito curto: temporização prematura
  - retransmissões são desnecessárias
- muito longo: reação demorada à perda de segmentos

P: como estimar RTT?

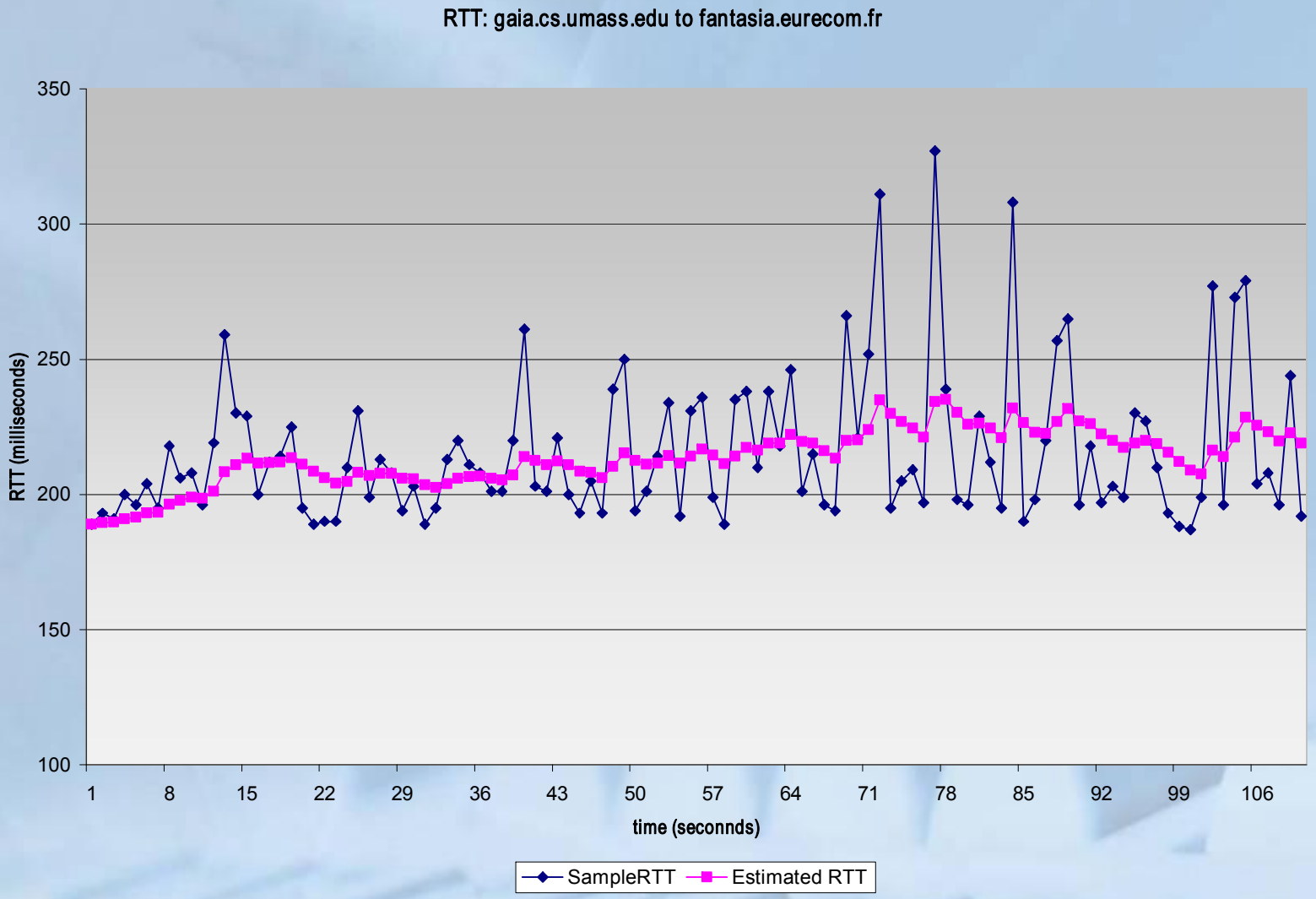
- `RTTamostra`: tempo medido entre a transmissão do segmento e o recebimento do ACK correspondente
  - ignora retransmissões
- `RTT_amostra` vai variar, queremos "amaciador" de RTT estimado
  - usa várias medições recentes, não apenas o valor corrente (`RTT_amostra`)

TCP: Tempo de Resposta (RTT) e Temporização

$$RTT\_estimado = (1-\alpha) * RTT\_estimado + \alpha * RTT\_amostra$$

- média corrente exponencialmente ponderada
- influência de cada amostra diminui exponencialmente com o tempo
- valor típico de α = 0,125

Exemplo de estimativa do RTT:



TCP: Tempo de Resposta (RTT) e Temporização

Escolhendo o intervalo de temporização

- `RTT_estimado` mais uma "margem de segurança"
  - grande variação no `RTT_estimado`
    -> maior margem de segurança
- primeiro estima o quanto a RTTamostra desvia do RTT_estimado:

$$Desvio\_RTT = (1-\beta) * Desvio\_RTT + \beta * |RTT\_amostra - RTT\_estimado|$$

- Então, seta o temporizador para:

$$Temporização = RTT\_estimado + 4 * Desvio\_RTT$$

Conteúdo do Capítulo 3

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 UDP: Transporte não orientado a conexão
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável
  - controle de fluxo
  - gerenciamento de conexões
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

Transferência de dados confiável do TCP

- O TCP cria um serviço rdt sobre o serviço não confiável do IP
- Segmentos em série (*pipelined)*
- Acks cumulativos
- O TCP usa um único temporizador para retransmissões

- As retransmissões são disparadas por:
  - estouros de temporização
  - acks duplicados
- Considere inicialmente um transmissor TCP simplificado:
  - ignora acks duplicados
  - ignora controles de fluxo e de congestionamento

Eventos do transmissor TCP:

Dados recebidos da apl.:

- Cria segmento com no. de seqüência (nseq)
- nseq é o número de seqüência do primeiro byte do segmento
- Liga o temporizador se já não estiver ligado (temporização do segmento mais antigo ainda não reconhecido)
- Valor do temporizador: calculado anteriormente

estouro do temporizador:

- Retransmite o segmento que causou o estouro do temporizador
- Reinicia o temporizador

Recepção de Ack:

- Se reconhecer segmentos ainda não reconhecidos
  - atualizar informação sobre o que foi reconhecido
  - religa o temporizador se ainda houver segmentos pendentes (não reconhecidos)

Transmis-sorTCP (simplificado)

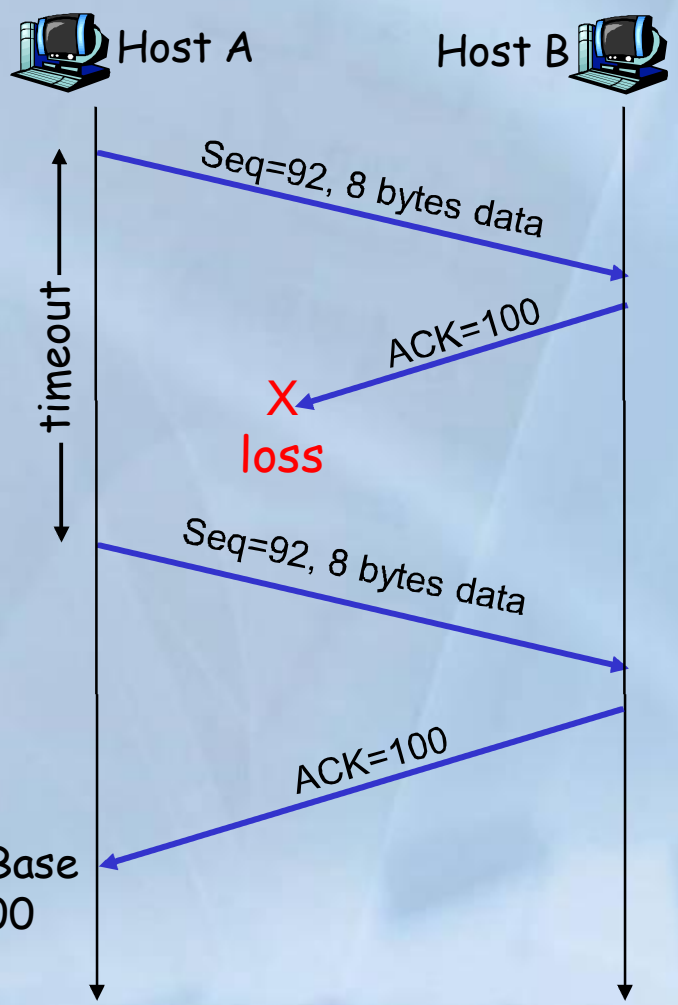
NextSeqNum = número de seqüência inicial
SendBase = número de seqüência inicial

```
repita (sempre) {
  switch(event)

  event: dados recebidos da aplicação acima
      cria segmento TCP com número de seqüência NextSeqNum
      se (temporizador estiver desligado)
           liga o temporizador
      passa segmento para IP
      NextSeqNum = NextSeqNum + comprimento(dados)

  event: estouro do temporizador
      retransmite segmento ainda não reconhecido com o
          menor número de seqüência
      reinicia o temporizador

  event: ACK recebido, com valor de campo ACK de y
      se (y > SendBase) { /* ACK cumulativo de todos dados até y */
          SendBase = y
          se (houver segmentos ainda não reconhecidos)
               liga o temporizador
          }  senão desliga o temporizador
} /* fim do repita sempre */
```

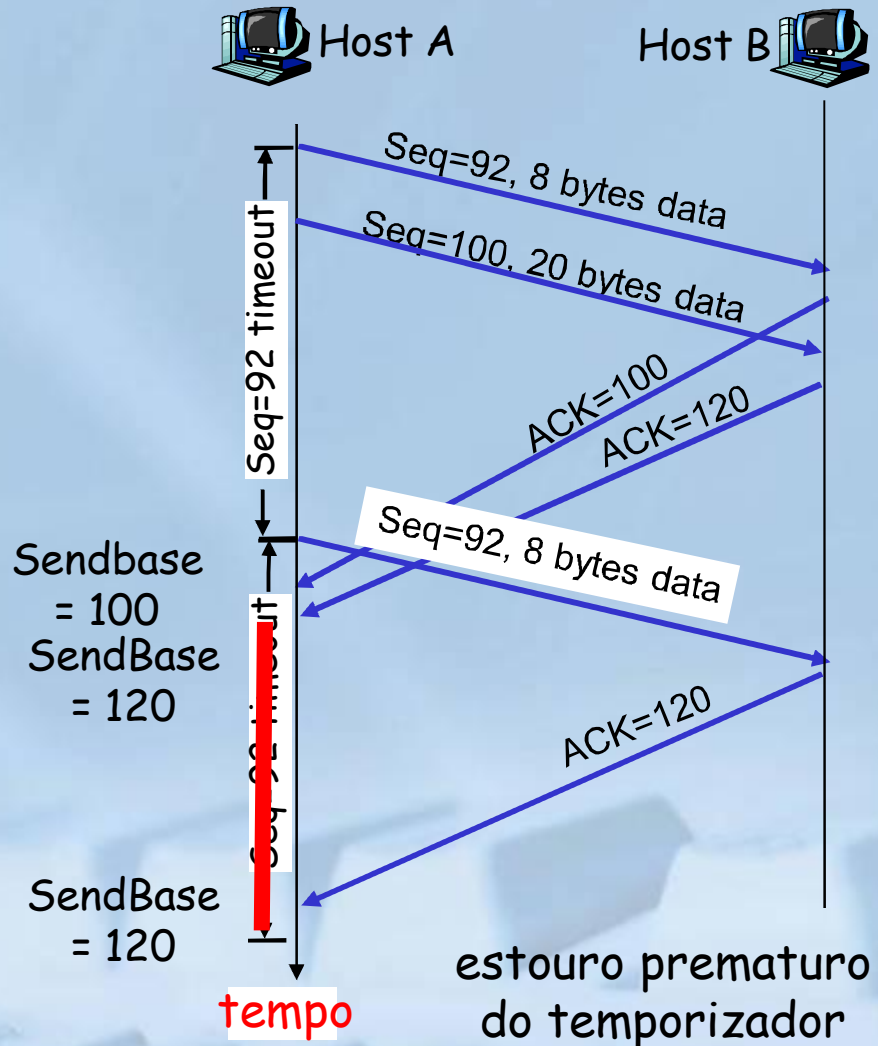
Comentário:
- SendBase-1: último byte reconhecido cumulativamente

Exemplo:
- SendBase-1 = 71; y= 73, portanto o receptor quer receber 73+;
- y > SendBase, portanto novos dados foram reconhecidos.

TCP: cenários de retransmissão

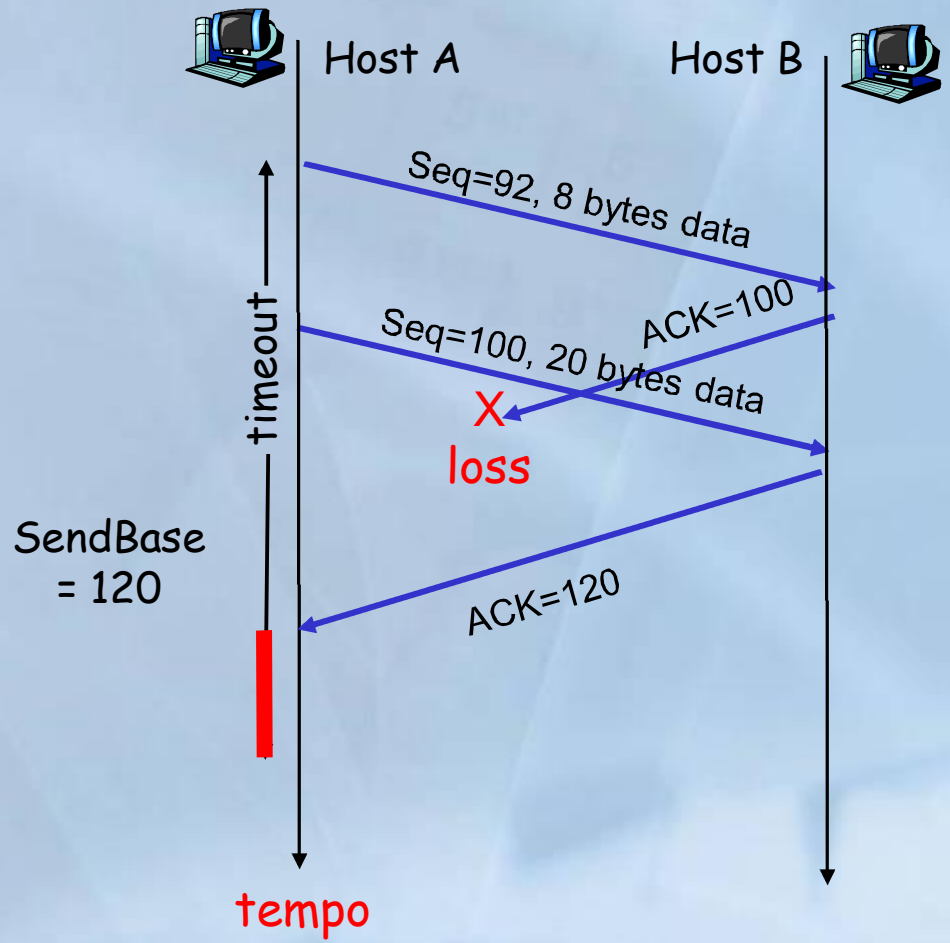


cenário de perda de ACK



estouro prematuro
do temporizador

TCP: cenários de retransmissão (mais)



Cenário de ACK cumulativo

TCP geração de ACKs [RFCs 1122, 2581]

Evento no Receptor	Ação do Receptor TCP
chegada de segmento em ordem sem lacunas, anteriores já reconhecidos	ACK retardado. Espera até 500ms p/ próx. segmento. Se não chegar segmento, envia ACK
chegada de segmento em ordem sem lacunas, um ACK retardado pendente	envia imediatamente um único ACK cumulativo
chegada de segmento fora de ordem, com no. de seq. maior que esperado -> lacuna	envia ACK duplicado, indicando no. de seq.do próximo byte esperado
chegada de segmento que preenche a lacuna parcial ou completamente	ACK imediato se segmento no início da lacuna

Retransmissão rápida

- O intervalo do temporizador é freqüentemente bastante longo:
  - longo atraso antes de retransmitir um pacote perdido
- Detecta segmentos perdidos através de ACKs duplicados.
  - O transmissor normalmente envia diversos segmentos
  - Se um segmento se perder, provavelmente haverá muitos ACKs duplicados.
- Se o transmissor receber 3 ACKs para os mesmos dados, ele supõe que o segmento após os dados reconhecidos se perdeu:
  - Retransmissão rápida: retransmite o segmento antes que estoure o temporizador

Algoritmo de retransmissão rápida:

```
event: recebido ACK, com valor do campo ACK de y
            if (y > SendBase) {
                 SendBase = y
                 if (houver segmentos ainda não reconhecidos)
                      liga temporizador
      else desliga temporizador
                 }
            else {
                 incrementa contador de ACKs duplicados recebidos para y
                 if (contador de ACKs duplicados recebidor para y = 3) {
                      retransmita segmento com número de seqüência y
                  }
```

um ACK duplicado para um segmento já reconhecido

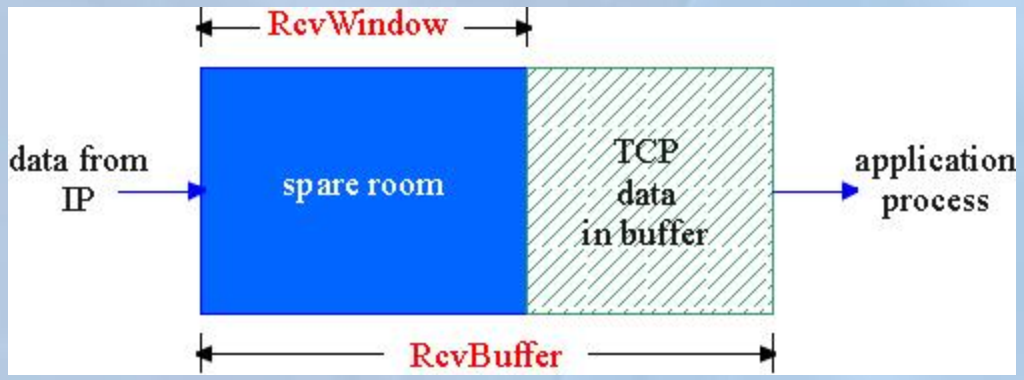
Retransmissão rápida

Conteúdo do Capítulo 3

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 UDP: Transporte não orientado a conexão
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável
  - controle de fluxo
  - gerenciamento de conexões
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

Controle de Fluxo do TCP

- Lado receptor da conexão TCP possui um buffer de recepção:



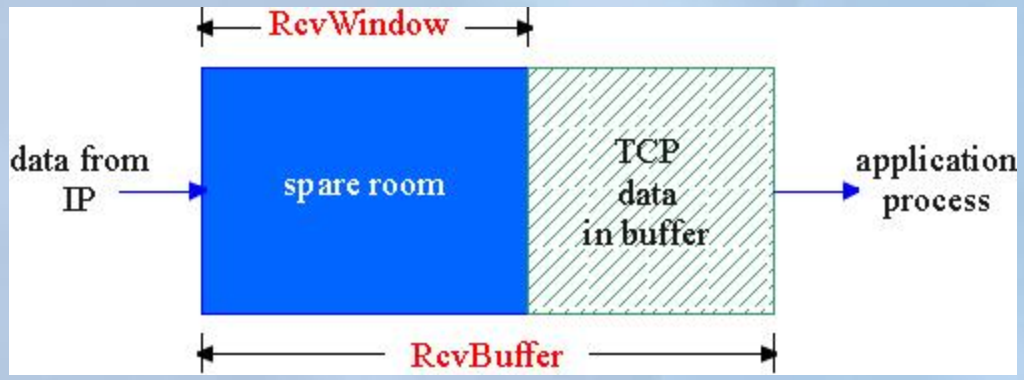
- Processo da apl. pode demorar a ler do receptor

Controle de fluxo

o transmissor não inundará o buffer do receptor transmitindo muito e rapidamente

- serviço de casamento de velocidades: adaptando a taxa de transmissão à taxa de leitura da aplicação receptora

Controle de Fluxo do TCP: como funciona



(Suponha que o receptor TCP segmentos fora de ordem)

- espaço livre no buffer

= `RcvWindow`

= `RcvBuffer-[LastByteRcvd - LastByteRead]`

- O receptor anuncia o espaço livre incluindo o valor da `RcvWindow` nos segmentos
- O transmissor limita os dados não reconhecidos ao tamanho da `RcvWindow`
  - Garante que o buffer do receptor não transbordará

Conteúdo do Capítulo 3

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 UDP: Transporte não orientado a conexão
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável
  - controle de fluxo
  - gerenciamento de conexões
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

TCP: Gerenciamento de Conexões

Lembrete: Remetente, receptor TCP estabelecem "conexão" antes de trocar segmentos de dados

- inicializam variáveis TCP:
  - nos. de seq.
  - buffers, info s/ controle de fluxo (p.ex. `RcvWindow`)
- *cliente:* iniciador de conexão

```
Socket clientSocket = new
 Socket("hostname","port
 number");
```

- *servidor:* contactado por cliente

```
Socket connectionSocket =
 welcomeSocket.accept();
```

Inicialização em 3 tempos:

Passo 1: sistema cliente envia segmento de controle SYN do TCP ao servidor
- especifica no. inicial de seq
- não envia dados

Passo 2: sistema servidor recebe SYN, responde com segmento de controle SYNACK
- aloca buffers
- especifica no. inicial de seq. servidor-> receptor

Passo 3: receptor recebe SYNACK, responde com segmento ACK que pode conter dados.

TCP: Gerenciamento de Conexões (cont.)

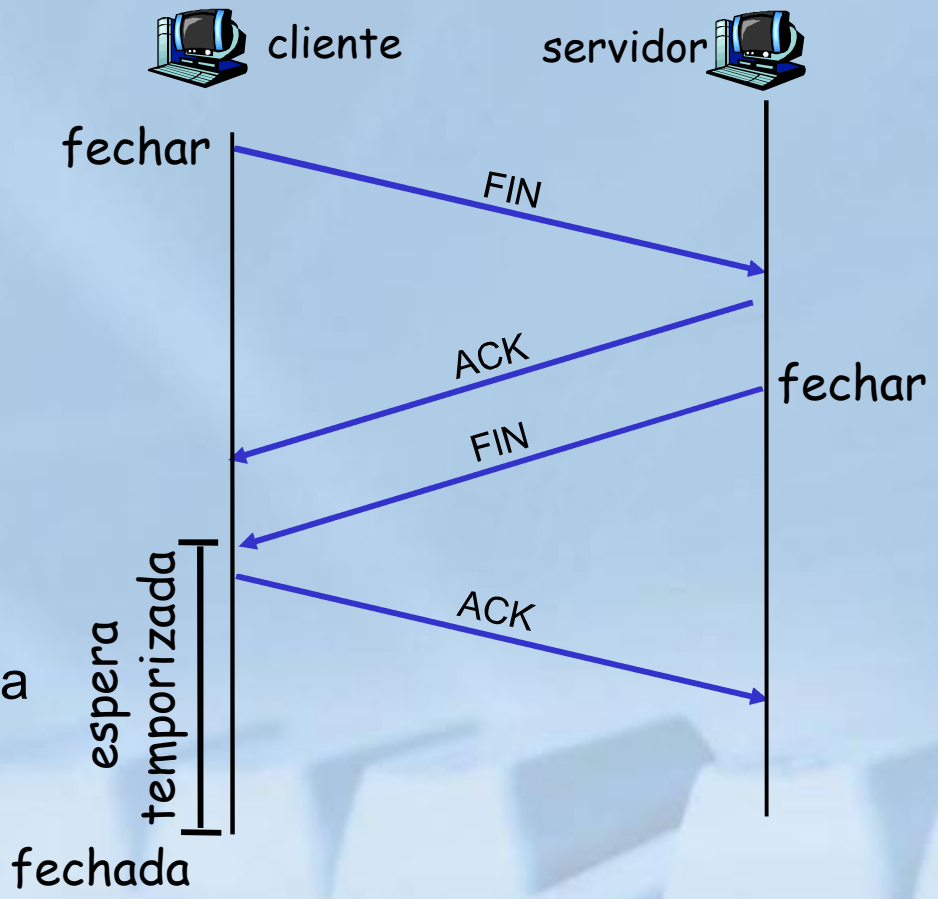
Encerrando uma conexão:

cliente fecha soquete:

```
clientSocket.close();
```

Passo 1: sistema cliente envia segmento de controle FIN ao servidor

Passo 2: servidor recebe FIN, responde com ACK. Encerra a conexão, enviando FIN.



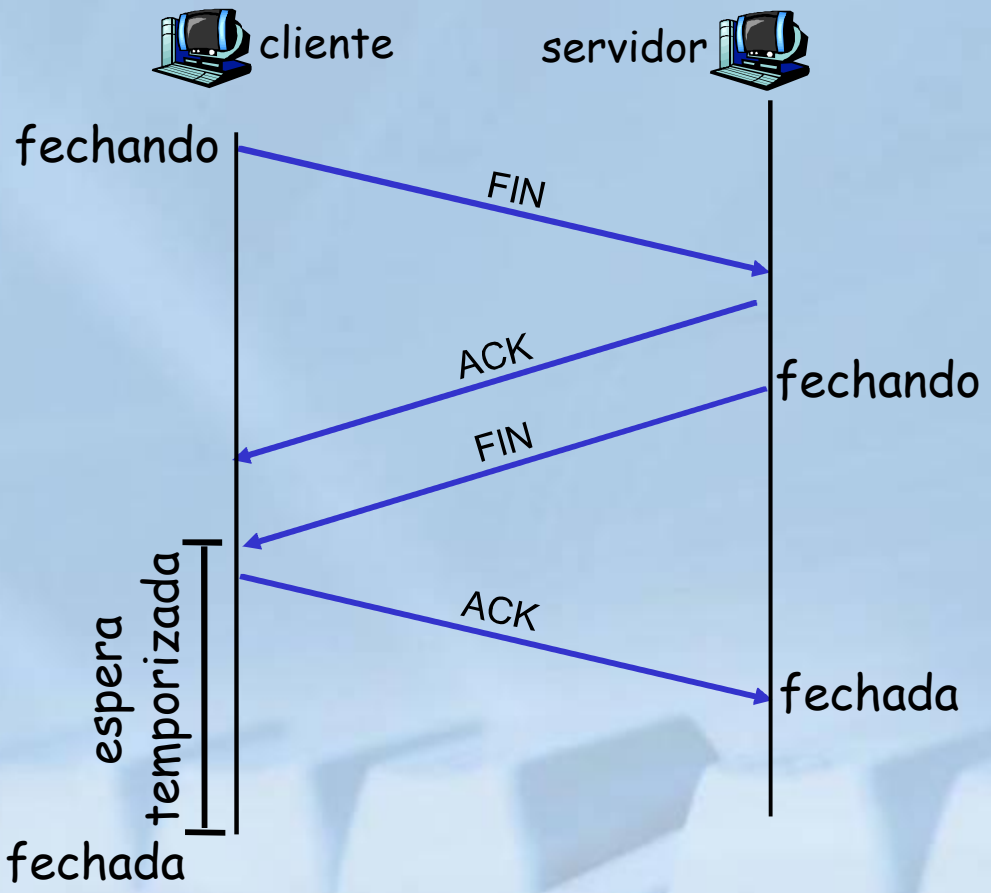
TCP: Gerenciamento de Conexões (cont.)

Passo 3: cliente recebe FIN, responde com ACK.

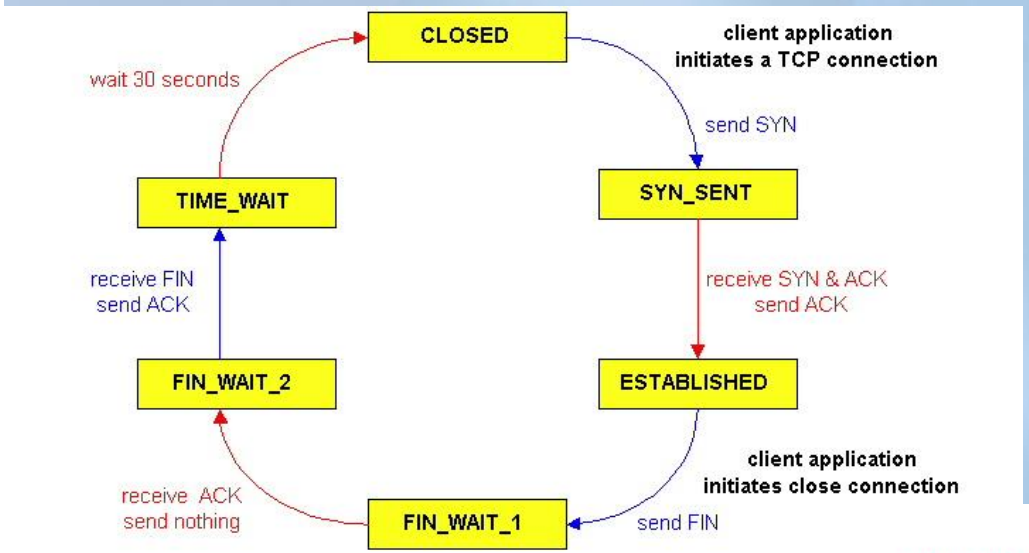
- Entre em "espera temporizada" - responderá com ACK a FINs recebidos

Passo 4: servidor, recebe ACK. Conexão encerrada.

Note: com pequena modificação, consegue tratar de FINs simultâneos.

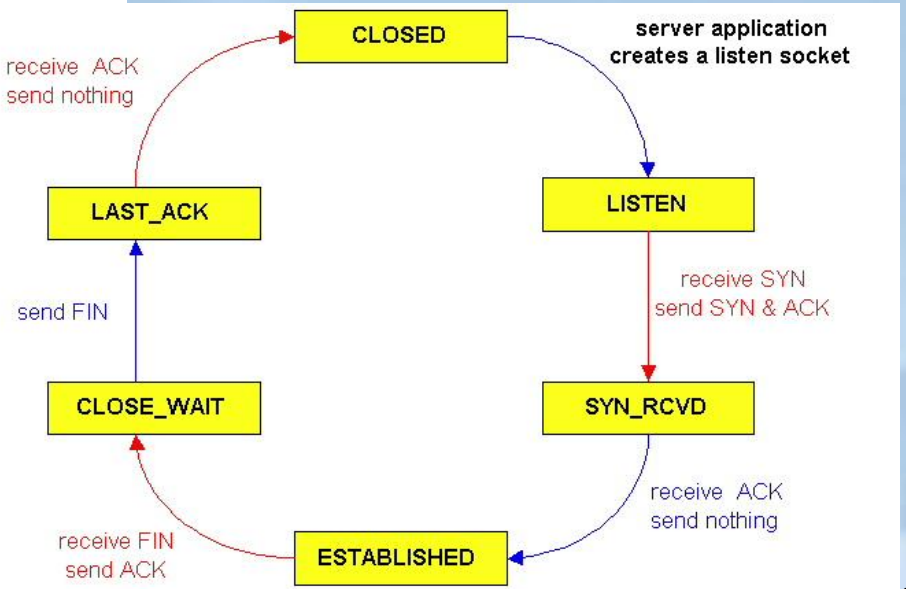


TCP: Gerenciamento de Conexões (cont.)



Ciclo de vida
de cliente TCP

Ciclo de vida
de servidor TCP



Conteúdo do Capítulo 3

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 UDP: Transporte não orientado a conexão
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável
  - controle de fluxo
  - gerenciamento de conexões
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

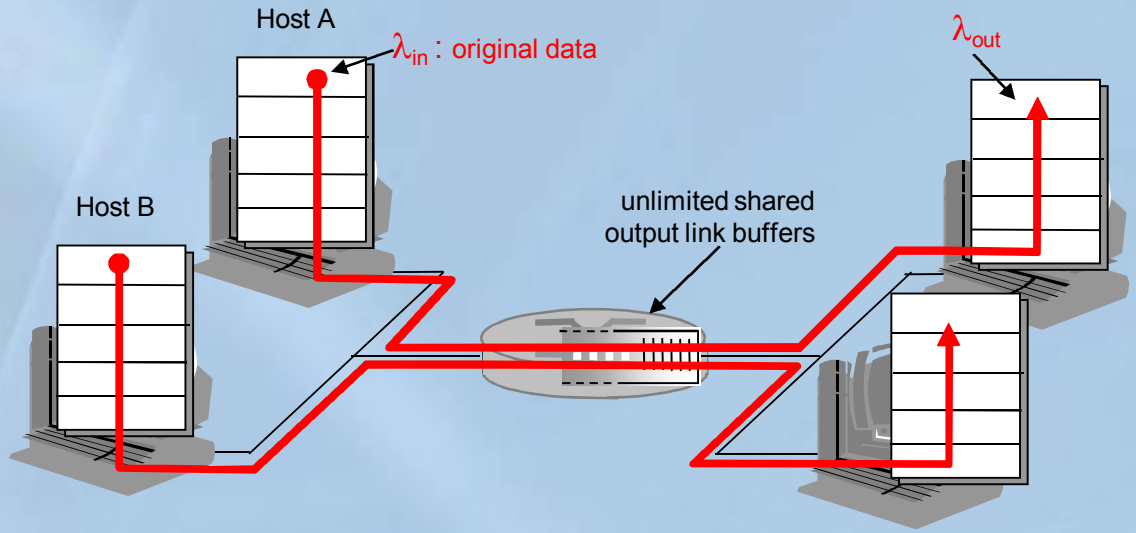
Princípios de Controle de Congestionamento

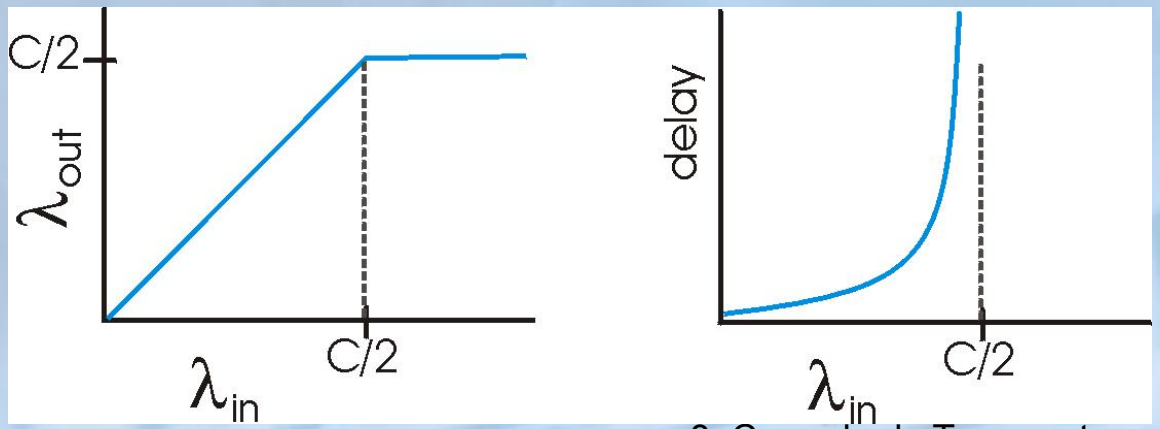
Congestionamento:

- informalmente: “muitas fontes enviando muitos dados muito rapidamente para a *rede* poder tratar”
- diferente de controle de fluxo!
- manifestações:
  - perda de pacotes (esgotamento de buffers em roteadores)
  - longos atrasos (enfileiramento nos buffers dos roteadores)
- um dos 10 problemas mais importantes em redes!

Causas/custos de congestionamento: cenário 1

- dois remetentes, dois receptores
- um roteador, *buffers* infinitos
- sem retransmissão

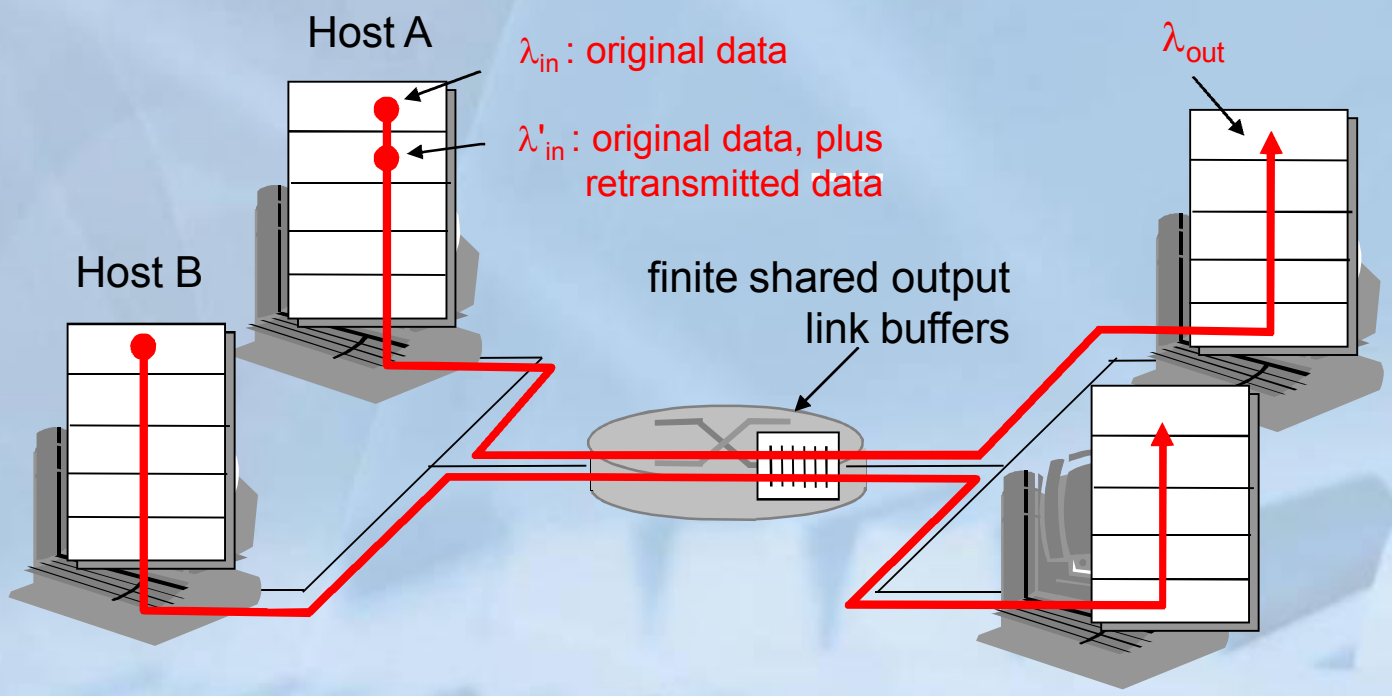




- grandes retardos qdo. congestionada
- vazão máxima alcançável

Causas/custos de congestionamento: cenário 2

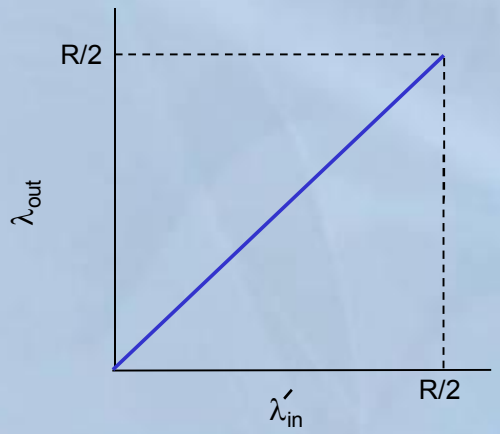
- Um roteador, buffers *finitos*
- retransmissão pelo remetente de pacote perdido



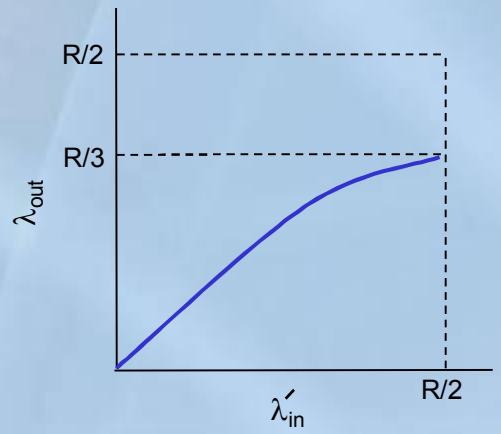
Causas/custos de congestionamento: cenário 2

$\lambda_{in} = \lambda_{out}$ sempre: (goodput)

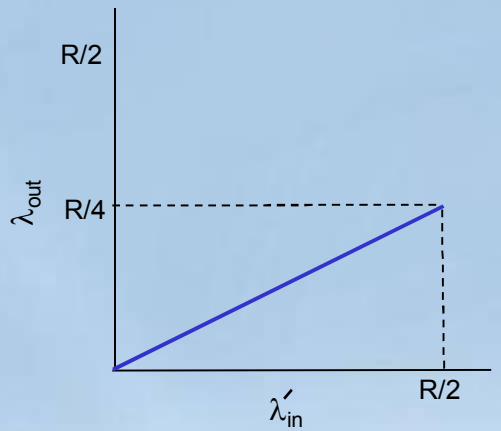
- retransmissão "perfeita" apenas com perdas: $\lambda'_{in} > \lambda_{out}$
- retransmissão de pacotes atrasados (não perdidos) torna λ'_{in} maior (do que o caso perfeito) para o mesmo λ_{out}



a.



b.



c.

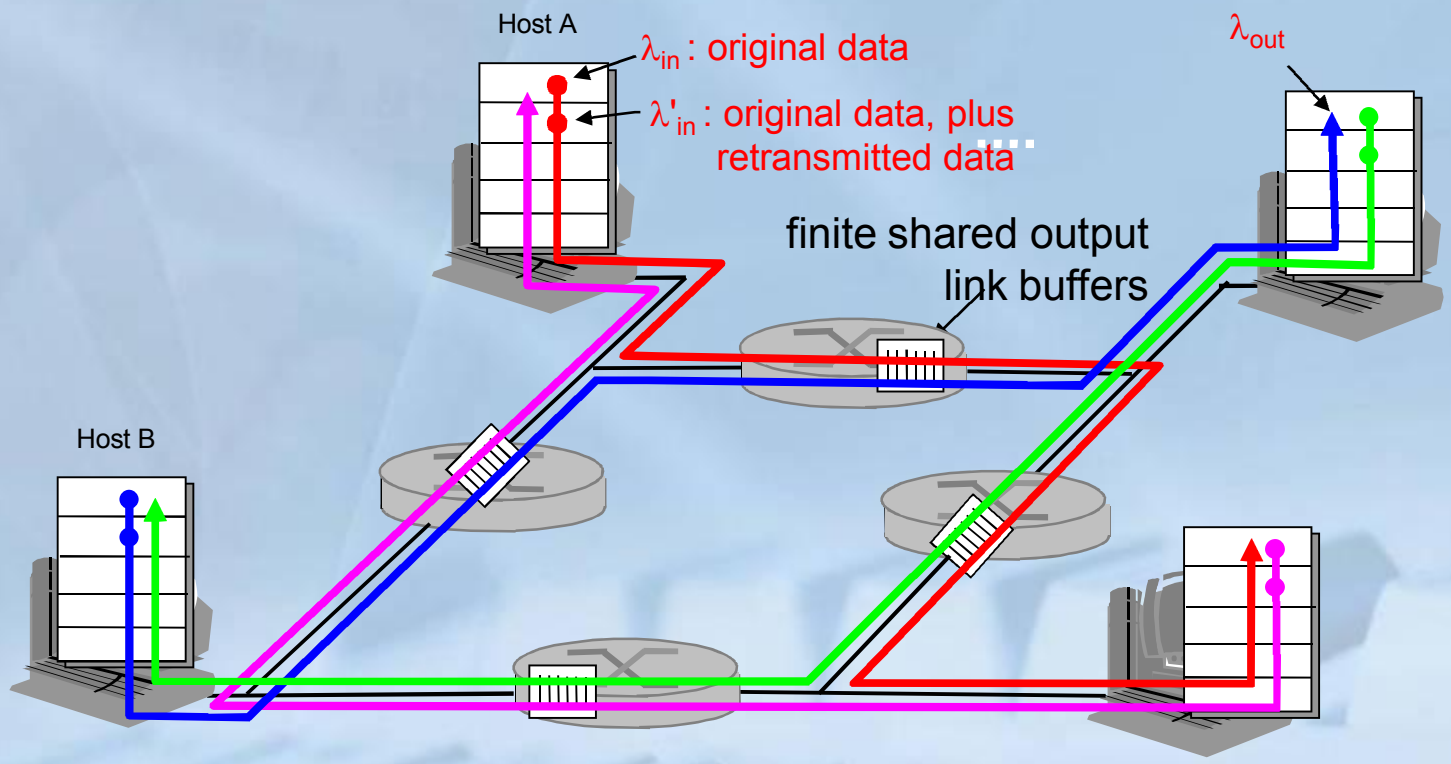
"custos" de congestionamento:

- mais trabalho (retransmissão) para dado "*goodput*"
- retransmissões desnecessárias: enviadas múltiplas cópias do pacote

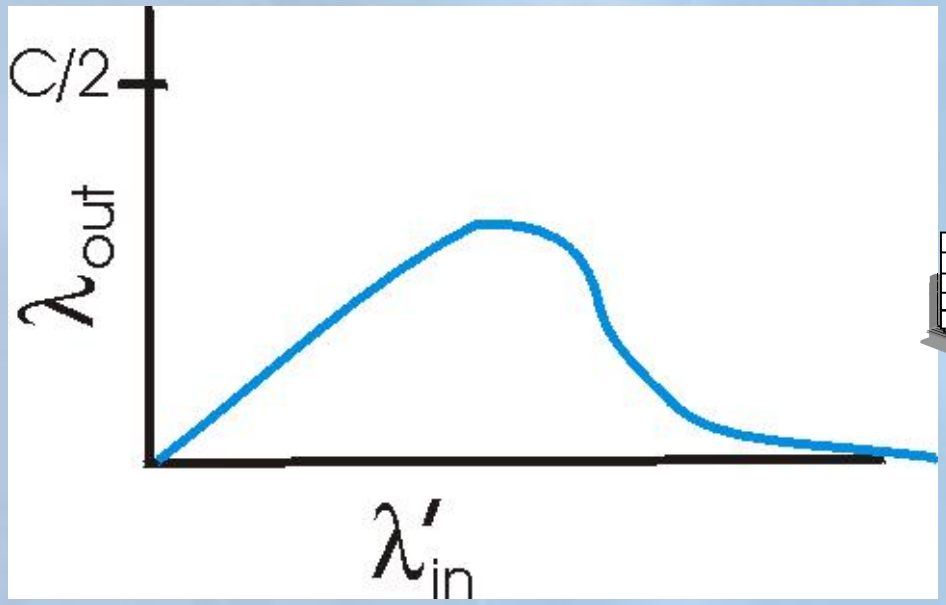
Causas/custos de congestionamento: cenário 3

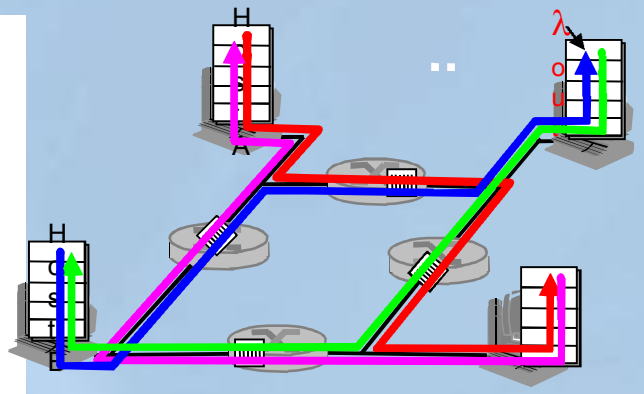
- quatro remetentes
- caminhos com múltiplos enlaces
- temporização/retransmissão

P: o que acontece à medida que λ_{in} e λ'_{in} crescem ?



Causas/custos de congestionamento: cenário 3





Outro "custo" de congestionamento:

- quando pacote é descartado, qq. capacidade de transmissão já usada (antes do descarte) para esse pacote foi desperdiçada!

Abordagens de controle de congestionamento

Duas abordagens amplas para controle de congestionamento:

Controle de congestionamento fim a fim :

- não tem realimentação explícita pela rede
- congestionamento inferido a partir das perdas, retardo observados pelo sistema terminal
- abordagem usada pelo TCP

Controle de congestionamento com apoio da rede:

- roteadores realimentam os sistemas terminais
  - bit indicando congestionamento (SNA, DECbit, TCP/IP ECN, ATM)
  - taxa explícita p/ envio pelo remetente

Estudo de caso: controle de congestionamento no ABR da ATM

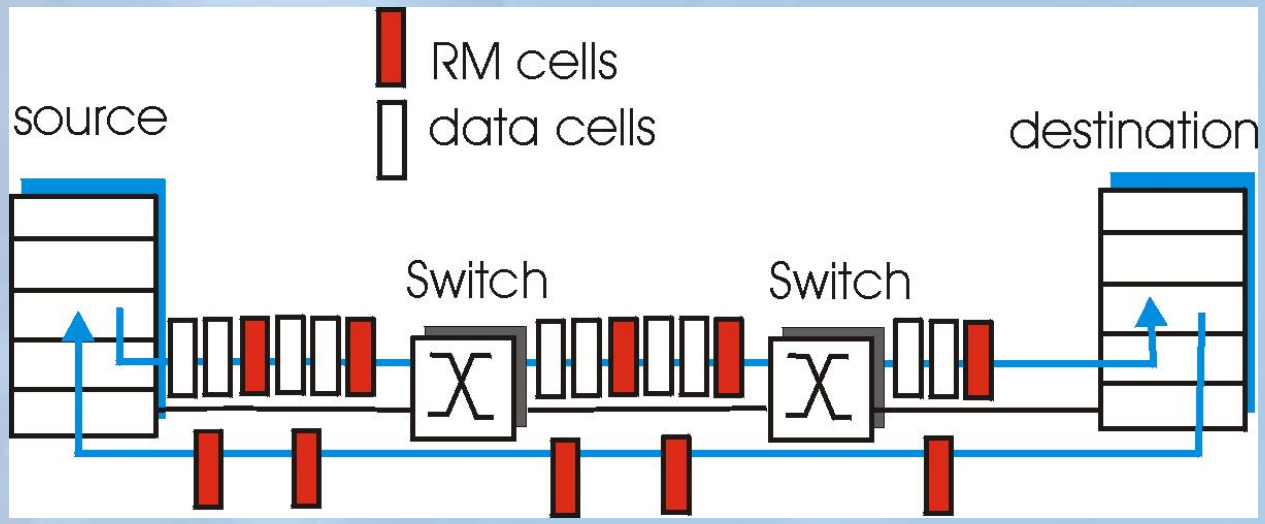
ABR (*available bit rate*):

- "serviço elástico"
- se caminho do remetente "subcarregado":
  - remetente deveria usar banda disponível
- se caminho do remetente congestionado:
  - remetente reduzido à taxa mínima garantida

células RM (*resource management*):

- enviadas pelo remetente, intercaladas com células de dados
- bits na célula RM iniciados por comutadores ("*apoio da rede*")
  - bit NI: não aumente a taxa (congestionamento moderado)
  - bit CI: indicação de congestionamento
- células RM devolvidas ao remetente pelo receptor, sem alteração dos bits

Estudo de caso: controle de congestionamento em ABR da ATM



- Campo ER (explicit rate) de 2 bytes na célula RM
  - comutador congestionado pode diminuir valor ER na célula
  - taxa do remetente assim ajustada p/ menor valor possível entre os comutadores do caminho
- bit EFCI em células de dados ligado por comutador congestionado
  - se EFCI ligado na célula de dados antes da célula RM, receptor liga bit CI na célula RM devolvida

Conteúdo do Capítulo 3

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 UDP: Transporte não orientado a conexão
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável
  - controle de fluxo
  - gerenciamento de conexões
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

Controle de Congestionamento do TCP

- controle fim-a-fim (sem assistência da rede)
- transmissor limita a transmissão:

  `LastByteSent-LastByteAcked`

  $\leq$ `CongWin`

- Praticamente,

$$\text{taxa} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/seg}$$

- `CongWin` é dinâmica, em função do congestionamento percebido da rede

Como o transmissor percebe o congestionamento?

- evento de perda = estouro do temporizador *ou* 3 acks duplicados
- transmissor TCP reduz a taxa (`CongWin`) após evento de perda

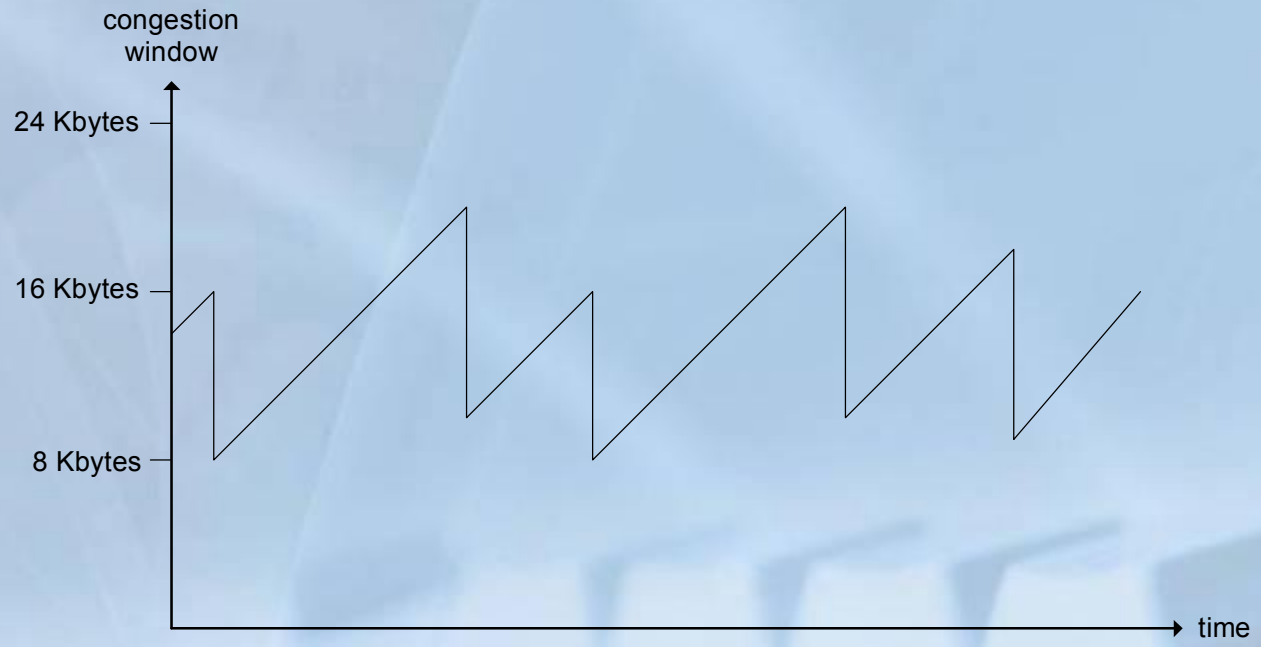
três mecanismos:

  - AIMD
  - partida lenta
  - conservador após eventos de estouro de temporização

AIMD do TCP

decrescimento multiplicativo: corta `CongWin` pela metade após evento de perda

crescimento aditivo: incrementa `CongWin` de 1 MSS a cada RTT na ausência de eventos de perda: *sondagem*



Conexão TCP de longa duração

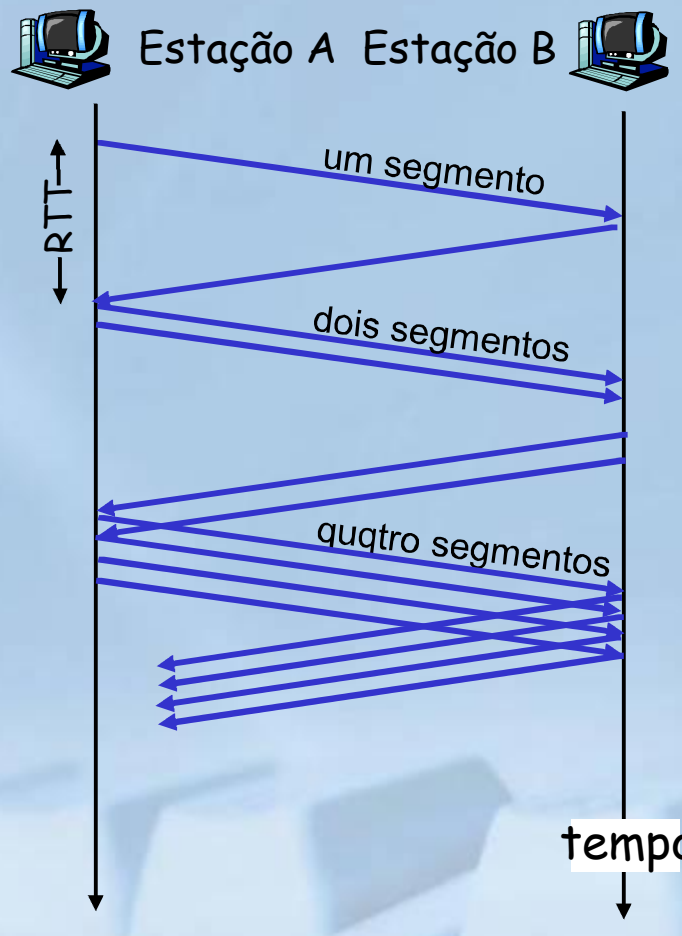
Partida Lenta do TCP

- No início da conexão, `CongWin` = 1 MSS
  - Exemplo: MSS = 500 bytes & RTT = 200 mseg
  - taxa inicial = 20 kbps
- largura de banda disponível pode ser >> MSS/RTT
  - é desejável um crescimento rápido até uma taxa considerável

- No início da conexão, aumenta a taxa exponencialmente até o primeiro evento de perda

TCP: Partida lenta (mais)

- No início da conexão, aumenta a taxa exponencialmente até o primeiro evento de perda:
  - duplica `CongWin` a cada RTT
  - através do incremento da `CongWin` para cada ACK recebido
- Resumo: taxa inicial é baixa mas cresce rapidamente de forma exponencial



Refinamento

- Após 3 ACKs duplicados:
  - corta `CongWin` pela metade
  - a janela depois cresce linearmente
- Mas após estouro de temporizador:
  - `CongWin` é reduzida a 1 MSS;
  - janela cresce exponencialmente
  - até um limiar, depois cresce linearmente

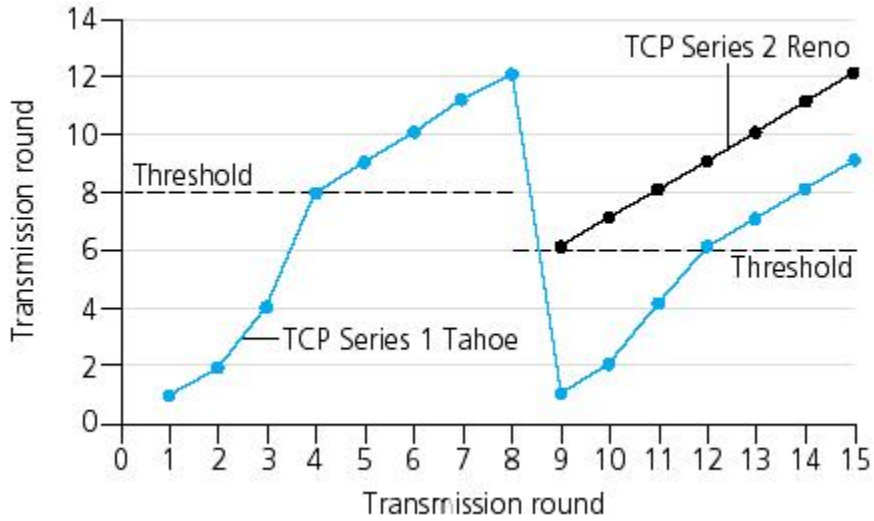
Filosofia:

- 3 ACKs duplicados indica que a rede é capaz de entregar alguns segmentos
- estouro de temporizador antes de 3 ACKs duplicados é mais "alarmante".

Refinamento (mais)

P: Quando o crescimento exponencial deve mudar para linear?

R: Quando `CongWin` atinge 1/2 do seu valor antes do estouro do temporizador.



Implementação:

- Limiar (*Threshold*) variável
- Com uma perda o limiar passa a ser 1/2 da CongWin imediatamente anterior à perda.

Resumo: Controle de Congestionamento do TCP

- Quando a `CongWin` está abaixo do `limiar`, transmissor está na fase de início lento, janela cresce exponencialmente.
- Quando a `CongWin` está acima do `limiar`, transmissor está na fase de evitar congestionamento, janela cresce linearmente.
- Quando chegam ACKs triplicados, `Limiar` passa a ser `CongWin/2` e `CongWin` passa ao valor do `Limiar`.
- Quando estoura o temporizador, `Limiar` passa a ser `CongWin/2` e `CongWin` passa a ser 1 MSS.

Controle de congestionamento do transmissor TCP

Evento	Estado	Ação do Transmissor TCP	Comentário
ACK recebido para dados ainda não reconhecidos	Partida lenta	CongWin = CongWin + MSS, If (CongWin > Limiar) seta estado para “Evitar congestionamento”	Resulta na duplicação da CongWin a cada RTT
ACK recebido para dados ainda não reconhecidos	Evitar congestion amento	CongWin = CongWin+MSS * (MSS/CongWin)	Incremento aditivo, resultando no incremento da CongWin de 1 MSS a cada RTT
Perda detectada por ACKs triplicados	qualquer	Limiar = CongWin/2, CongWin = Limiar, Seta estado para “Evitar Congestionamento”	Recuperação rápida, implementa decrescimento multiplicativo. CongWin não cai abaixo de 1 MSS.
Estouro de temporizador	qualquer	Limiar = CongWin/2, CongWin = 1 MSS, Seta estado para “Partida lenta”	Entra estado de “partida lenta”
ACK duplicado	qualquer	Incrementa contador de ACKs duplicados para o segmento que está sendo reconhecido	CongWin e Threshold não se alteram

Vazão (*throughput*) do TCP

- Qual é a vazão média do TCP em função do tamanho da janela e do RTT?
  - Ignore a partida lenta
- Seja W o tamanho da janela quando ocorre a perda
- Quando a janela é W a vazão é W/RTT
- Imediatamente após a perda, janela cai a W/2, vazão cai para W/2RTT.
- Vazão média = 0,75 W/RTT

Futuro do TCP

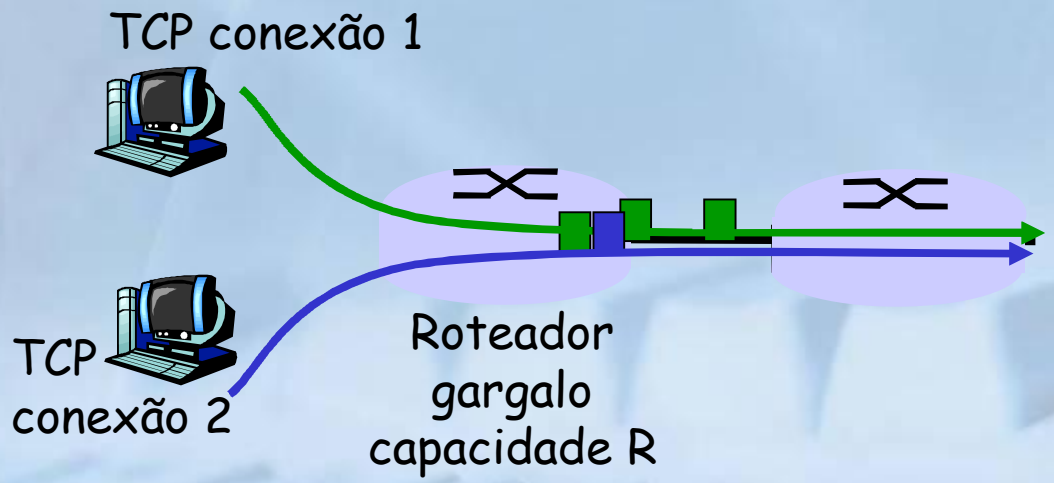
- Exemplo: segmentos de 1500 bytes, RTT de 100ms, deseja vazão de 10 Gbps
- Requer janela de W = 83.333 segmentos em trânsito
- Vazão em termos de taxa de perdas:

$$\frac{1{,}22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ L = $2 \cdot 10^{-10}$ *Taxa de perdas demasiado baixa!!!*
- São necessárias novas versões do TCP para altas velocidades!

Equidade (*Fairness*) do TCP

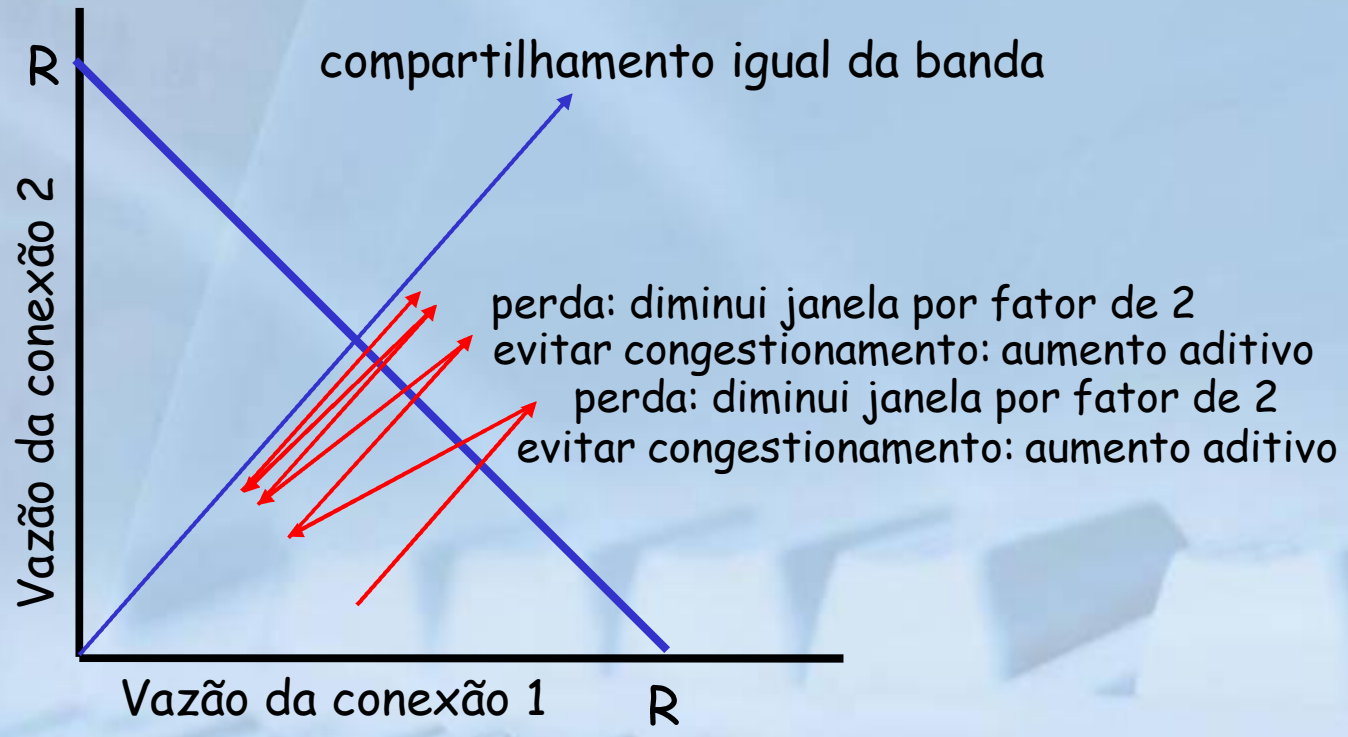
Meta de equidade: se K sessões TCP compartilham o mesmo enlace de gargalo com largura de banda R, cada uma deve obter uma taxa média de R/K



Por quê TCP é justo?

Duas sessões concorrentes:

- Aumento aditivo dá gradiente de 1, enquanto vazão aumenta
- decremento multiplicativo diminui vazão proporcionalmente



Justeza (mais)

Justeza e UDP

- Apls. multimídia freqüentemente não usam TCP
  - não desperdiçam taxa regulada pelo controle de congestionamento
- Preferem usar o UDP:
  - Injeta áudio/vídeo a uma taxa constante, tolera perda de pacotes
- Área de Pesquisa: amigável ao TCP (TCP *friendly*)

Justeza e conexões TCP em paralelo

- nada impede que as apls. abram conexões paralelas entre 2 hosts
- Os *browsers* Web fazem isto
- Exemplo: canal com taxa R compartilhado por 9 conexões;
  - nova apl pede 1 TCP, recebe taxa R/10
  - nova apl pede 11 TCPs, recebe taxa R/2 !

TCP: modelagem de latência

P: Quanto tempo leva para receber um objeto de um servidor WWW depois de enviar o pedido?

- Estabelecimento de conexão TCP
- retardo de transferência de dados

Notação, suposições:

- Supomos um enlace entre cliente e servidor de taxa R
- Supomos: janela de congestionamento fixo, W segmentos
- S: MSS (bits)
- O: tamanho do objeto (bits)
- sem retransmissões (sem perdas, sem erros)

Dois casos a considerar:

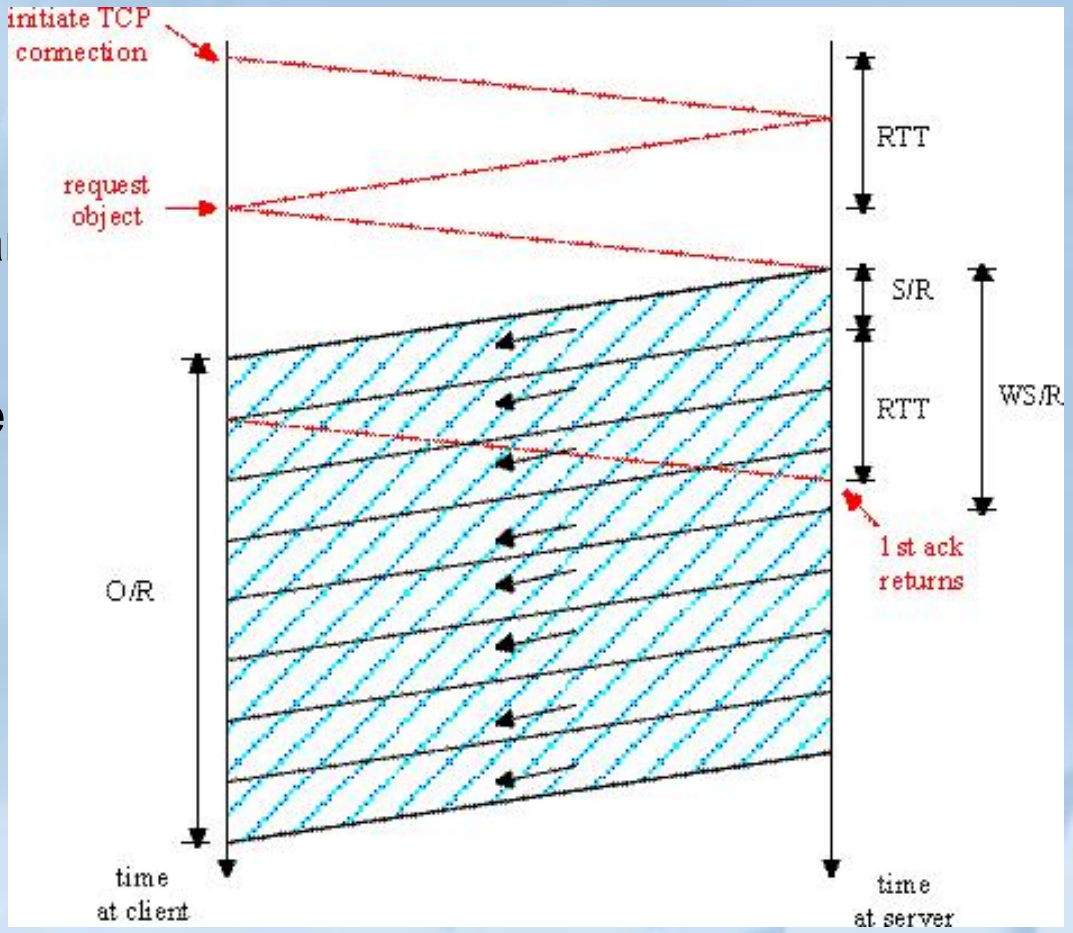
- WS/R > RTT + S/R: ACK do primeiro segmento na janela chega antes de enviar todos dados na janela
- WS/R < RTT + S/R: aguarda ACK depois de enviar todos os dados na janela

Janela de congestionamento fixa (1)

Primeiro caso:

WS/R > RTT + S/R: ACK pa
o primeiro segmento na janela retorna antes da transmissão de uma jane completa de dados

latência = 2RTT + O/R

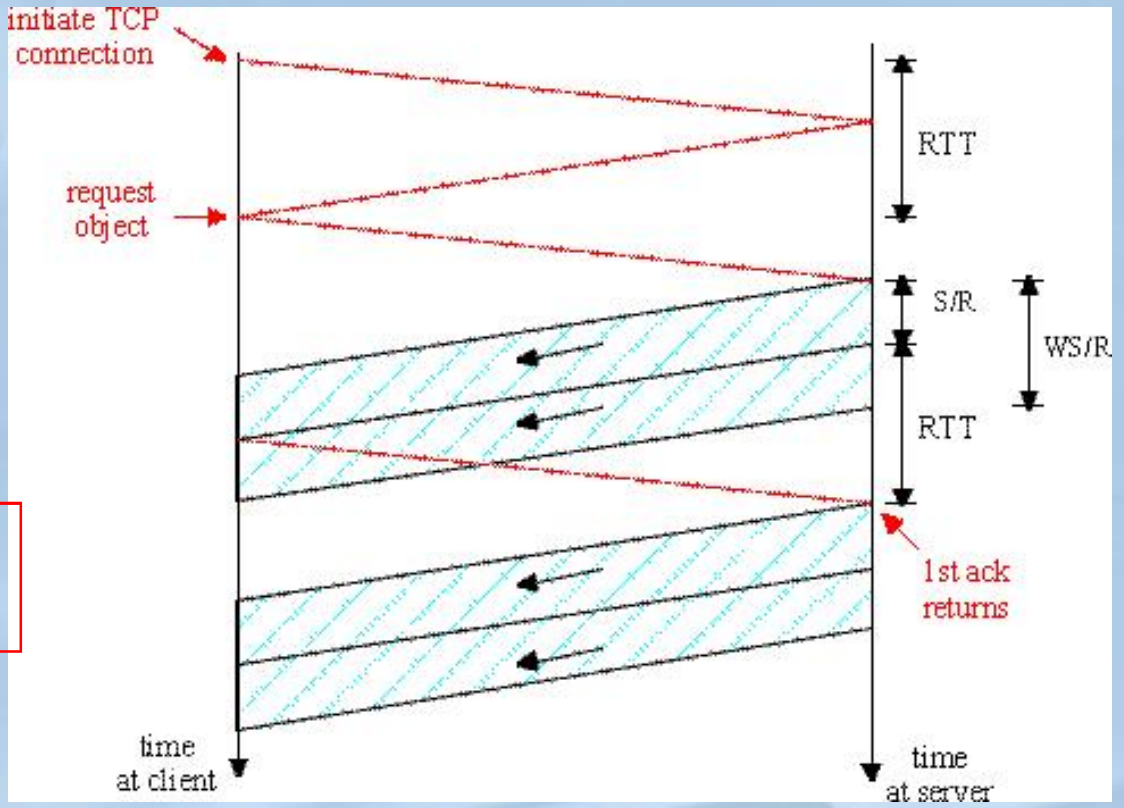


Janela de congestionamento fixa (2)

Segundo caso:

- WS/R < RTT + S/R: espera por ACK após transmitir uma janela completa de dados

latência = 2RTT + O/R
+ (K-1)[S/R + RTT - WS/R]



TCP: modelagem de latência: partida lenta

- Agora supomos que a janela cresce à la partida lenta.
- Mostramos que a latência de um objeto de tamanho O é:

$$Latência = 2RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - (2^P - 1)\frac{S}{R}$$

onde *P* é o número de vezes que o TCP para no servidor:

$$P = \min\{Q, K-1\}$$

- onde Q é o número de vezes que o servidor pararia se o objeto fosse de tamanho infinito.

- e K é o número de janelas que cobrem o objeto.

TCP: modelagem de latência: partida lenta (cont.)

Componentes da latência:

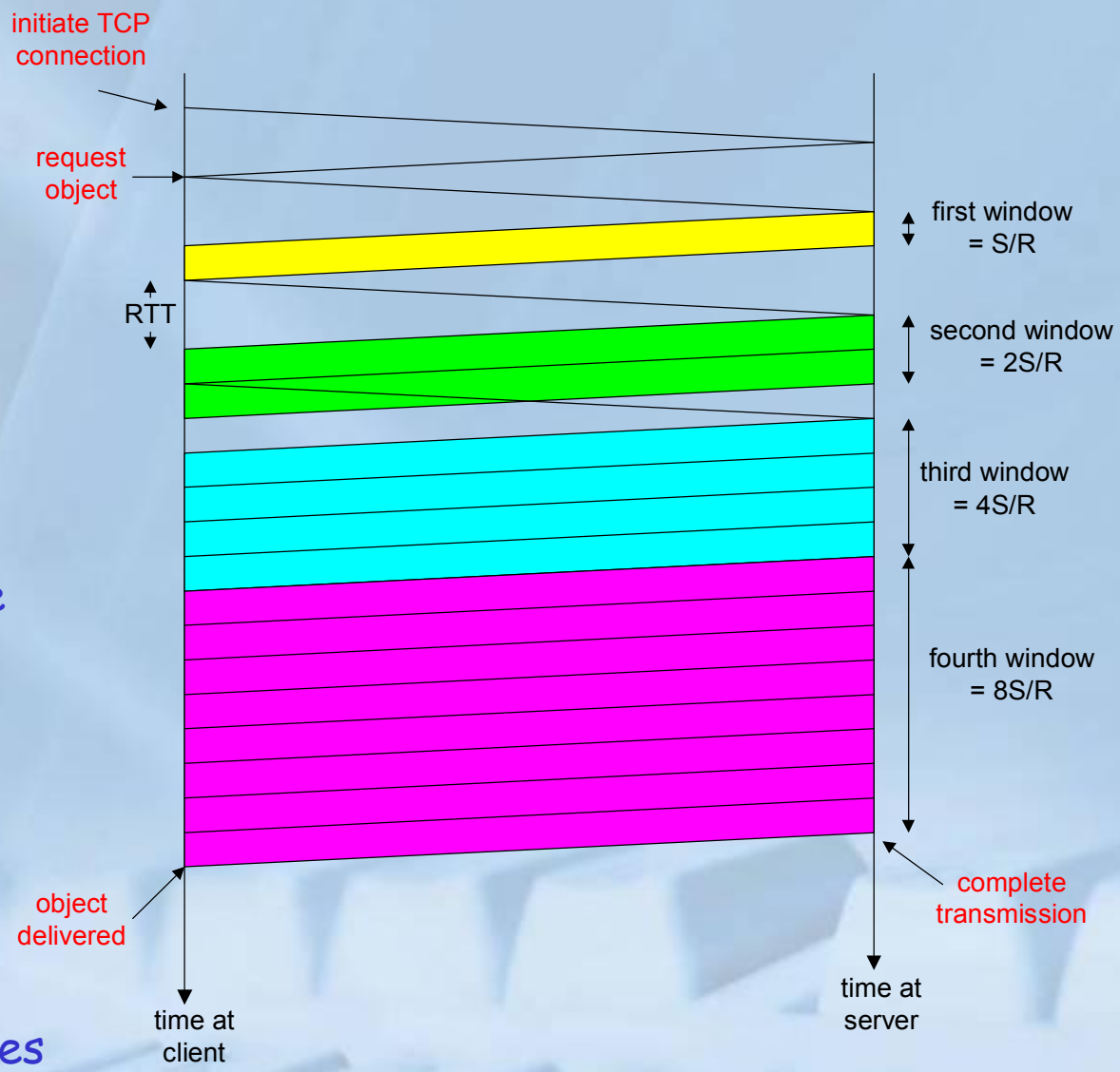
- 2 RTTs para estab conexão e pedido
- O/R para transmitir o objeto
- tempo ocioso do servidor devido à partida lenta

Servidor ocioso:
P = min{K-1,Q} unidades de tempo

Exemplo:

- O/S = 15 segmentos
- K = 4 janelas
- Q = 2
- P = min{K-1,Q} = 2

Servidor ocioso P=2 unidades de tempo



TCP: modelagem de latência: partida lenta (cont.)

$\frac{S}{R} + RTT =$ tempo desde que o servidor começa a enviar segmentos

até que o servidor receba os reconhecimentos

$2^{k-1}\frac{S}{R} =$ tempo para transmitir a k-ésima janela

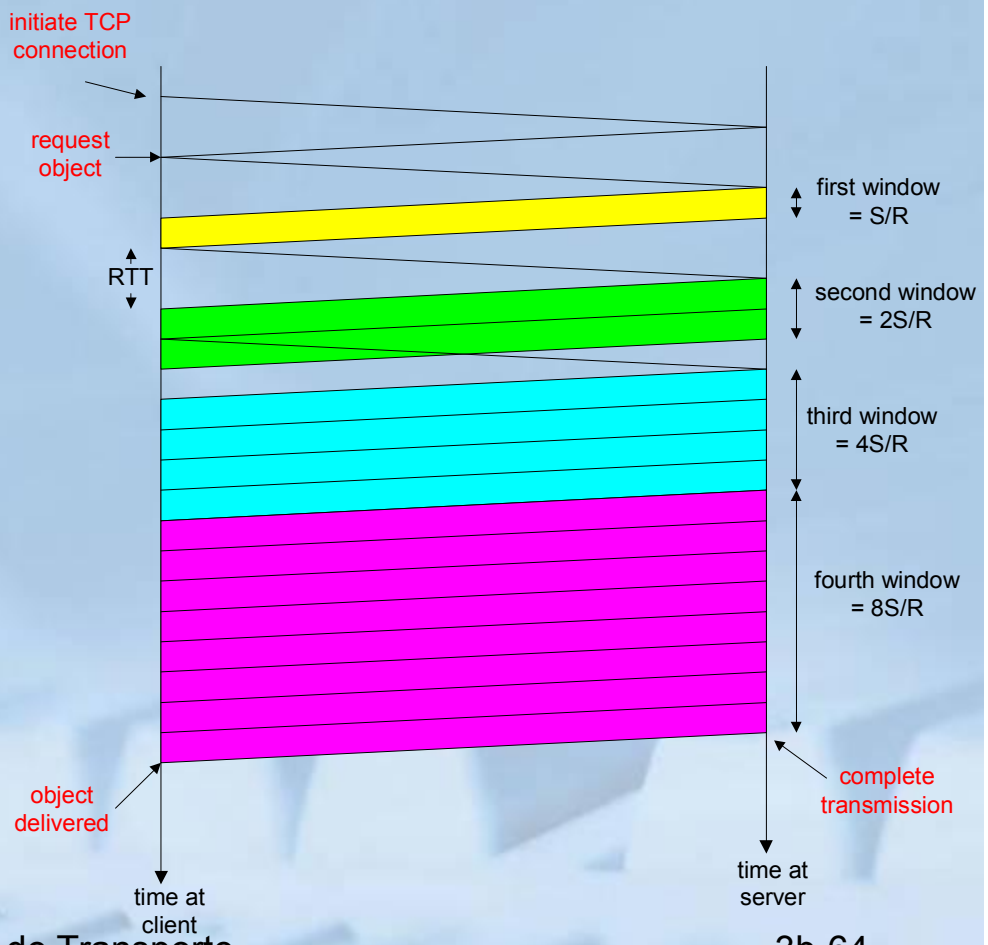
$\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^{+} =$

tempo parado após a $k-$ésima janela

$$\text{latência} = \frac{O}{R} + 2RTT + \sum_{p=1}^{P} Tempoparado_p$$

$$= \frac{O}{R} + 2RTT + \sum_{k=1}^{P}\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]$$

$$= \frac{O}{R} + 2RTT + P\left[RTT + \frac{S}{R}\right] - (2^{P} - 1)\frac{S}{R}$$



Modelagem de Latência do TCP (4)

Lembre que K = número de janelas que cobrem objeto
Como podemos calcular K ?

$$
\begin{aligned}
K &= \min\{k : 2^0 S + 2^1 S + \cdots + 2^{k-1} S \geq O\} \\
&= \min\{k : 2^0 + 2^1 + \cdots + 2^{k-1} \geq O / S\} \\
&= \min\{k : 2^k - 1 \geq \frac{O}{S}\} \\
&= \min\{k : k \geq \log_2(\frac{O}{S} + 1)\} \\
&= \left\lceil \log_2(\frac{O}{S} + 1) \right\rceil
\end{aligned}
$$

Cálculo de Q, número de intervalos ociosos para um objeto de tamanho infinito é semelhante (veja exercício).

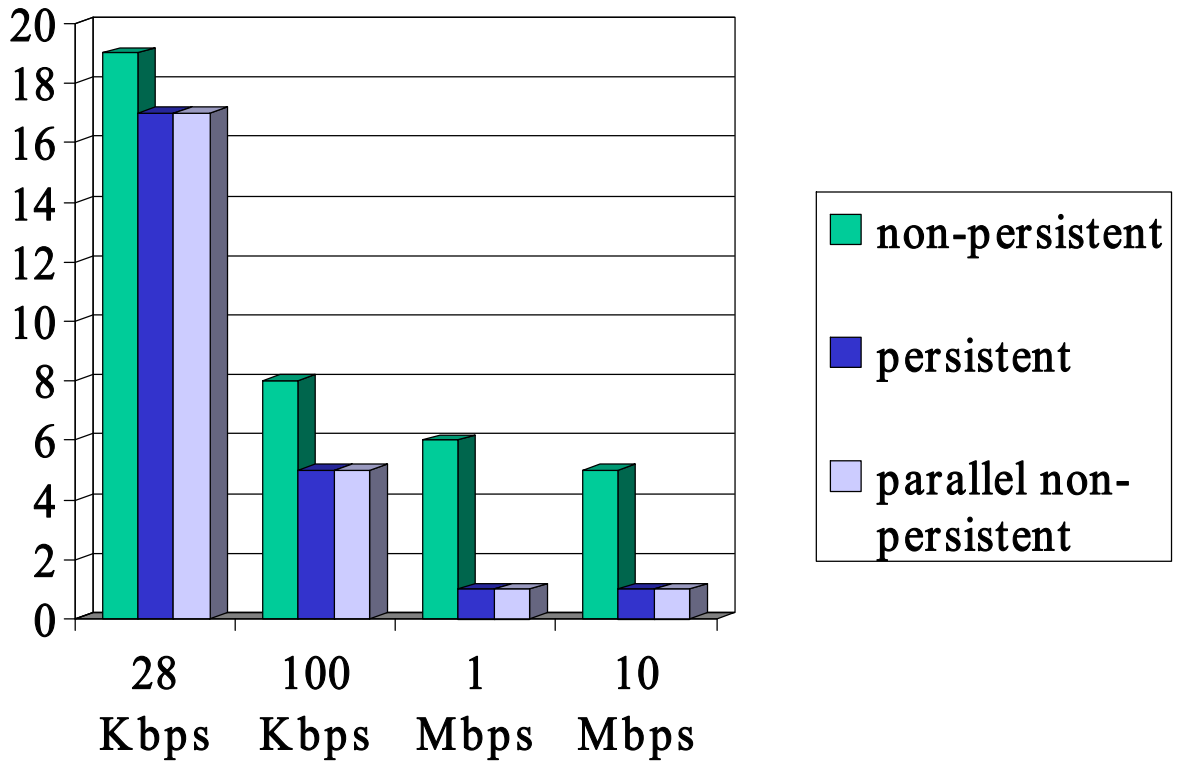
Modelagem do HTTP

- Assuma que a página Web é composta por:
  - *1* página base HTML (de tamanho *O* bits)
  - *M* imagens (cada uma de tamanho *O* bits)
- HTTP não-persistente :
  - *M+1* connexões TCP em série
  - *Tempo de resposta = (M+1)O/R + (M+1)2RTT + soma dos tempos ociosos*
- HTTP persistente :
  - *2 RTT* para pedir e receber arquivo HTML
  - *1 RTT* para pedir e receber M imagens
  - *Tempo de resposta = (M+1)O/R + 3RTT + soma dos tempos ociosos*
- HTTP não-persistente com X conexões paralelas
  - Suponha que M/X seja um inteiro.
  - 1 conexão TCP para arquivo base
  - Conjuntos de M/X conexões paralelas para as imagens.
  - *Tempo de resposta = (M+1)O/R + (M/X + 1)2RTT + soma dos tempos ociosos*

Tempo de resposta do HTTP

RTT = 100 mseg, O = 5 Kbytes, M=10 and X=5

em segundos



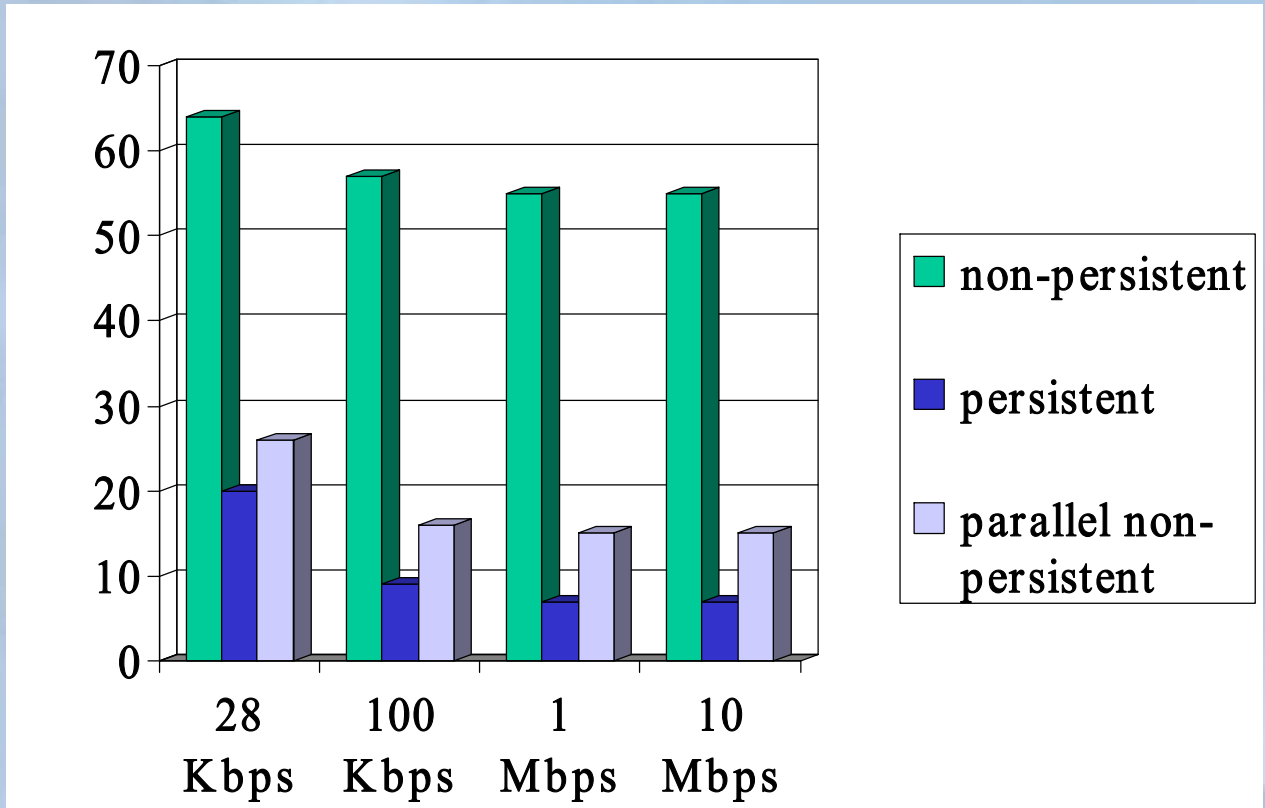
Para largura de banda baixa os tempos de conexão e de resposta são dominados pelo tempo de transmissão.

Conexões persistentes resultam num melhoramento pequeno em relação às conexões paralelas.

Tempo de resposta do HTTP

RTT =1 sec, O = 5 Kbytes, M=10 e X=5

em segundos



Para grandes RTTs, o tempo de resposta é dominado pelos atrasos de estabelecimento de conexões e de partida lenta.
Conexões persistentes apresentam um melhor desempenho: particularmente em redes com valor alto do produto atraso * largura de banda.

Capítulo 3: Resumo

- Princípios atrás dos serviços da camada de transporte:
  - multiplexação/ demultiplexação
  - transferência confiável de dados
  - controle de fluxo
  - controle de congestionamento
- instanciação e implementação na Internet
  - UDP
  - TCP

Próximo capítulo:

- saímos da "borda" da rede (camadas de aplicação e transporte)
- entramos no "núcleo"da rede