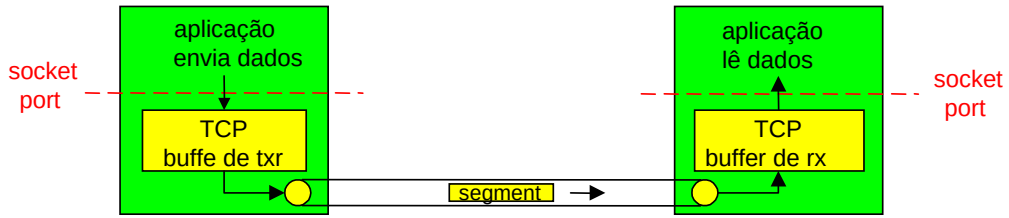
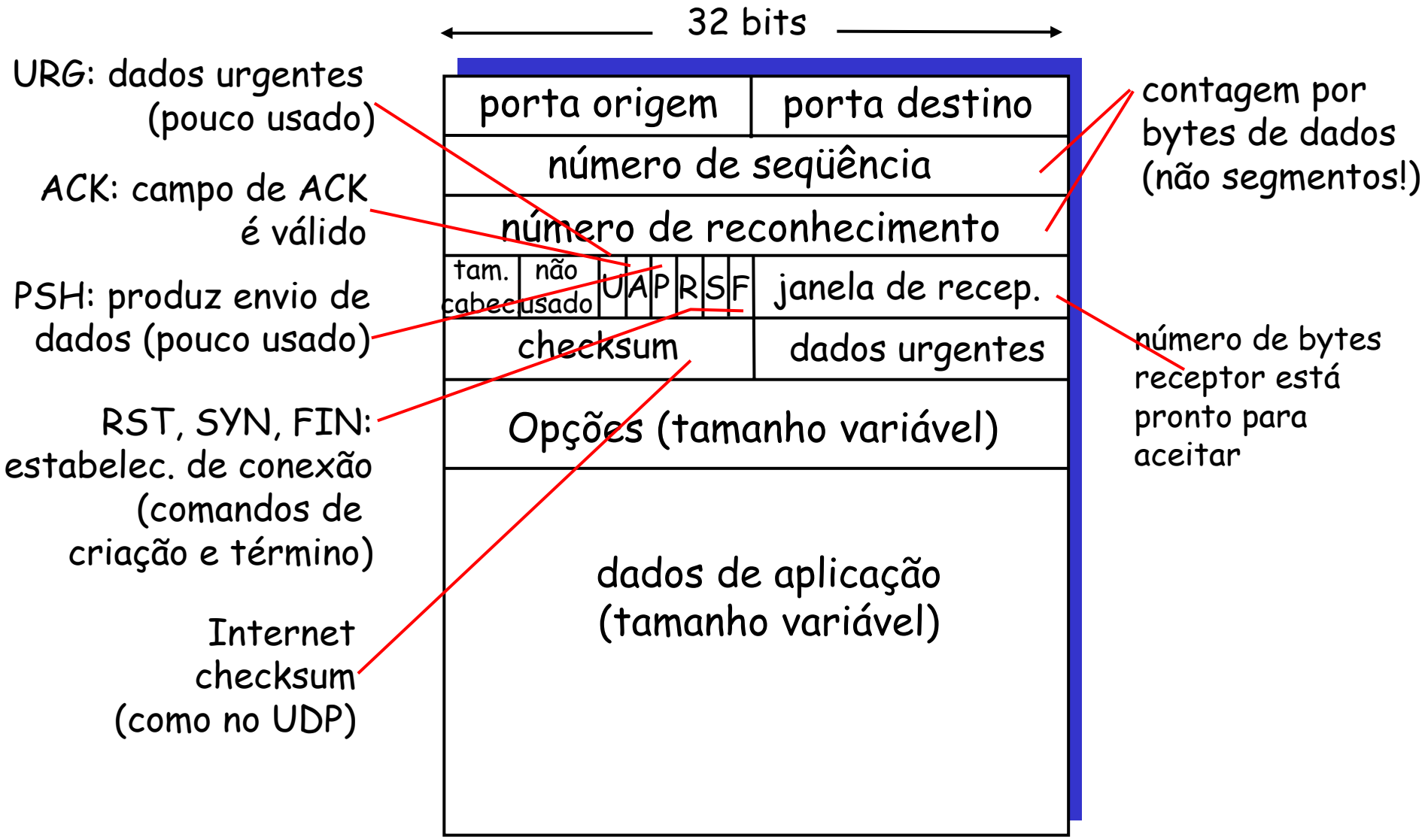
TCP: Visão Geral RFCs: 793, 1122, 1323, 2018, 2581

- ponto-a-ponto:
  - um transmissor, um receptor
- confiável, seqüêncial *byte stream:*
  - não há contornos de mensagens
- pipelined: (transmissão de vários pacotes em confirmação)
  - Controle de congestão e de fluxo definem tamanho da janela
- *buffers de transmissão e de recepção*



- dados full-duplex:
  - transmissão bi-direcional na mesma conexão
  - MSS: maximum segment size
- orientado à conexão:
  - handshaking (troca de mensagens de controle) inicia o estado do transmissor e do receptor antes da troca de dados
- controle de fluxo:
  - transmissor não esgota a capacidade do receptor

Estrutura do Segmento TCP



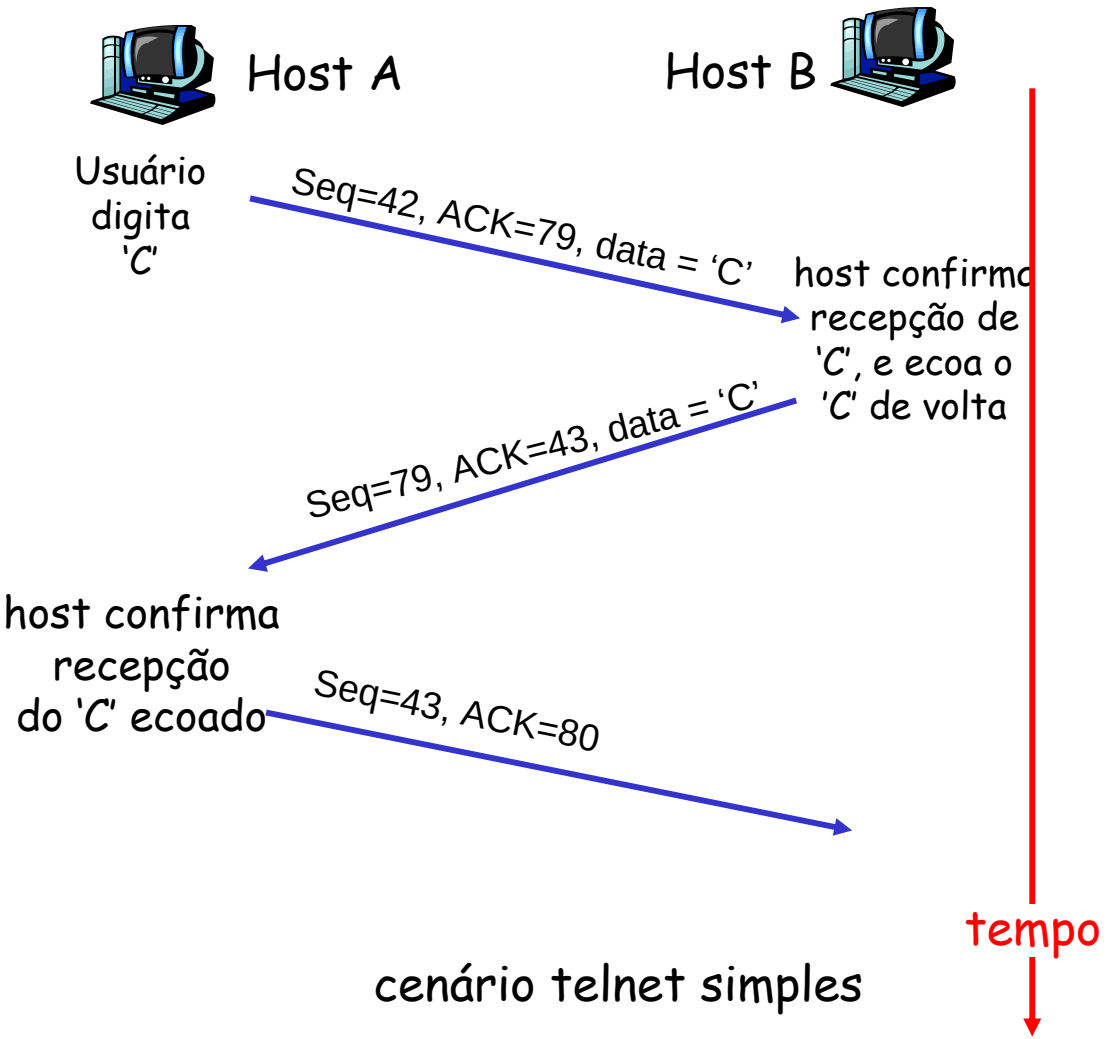
Números de Seqüência e ACKs do TCP

Números de seqüência:

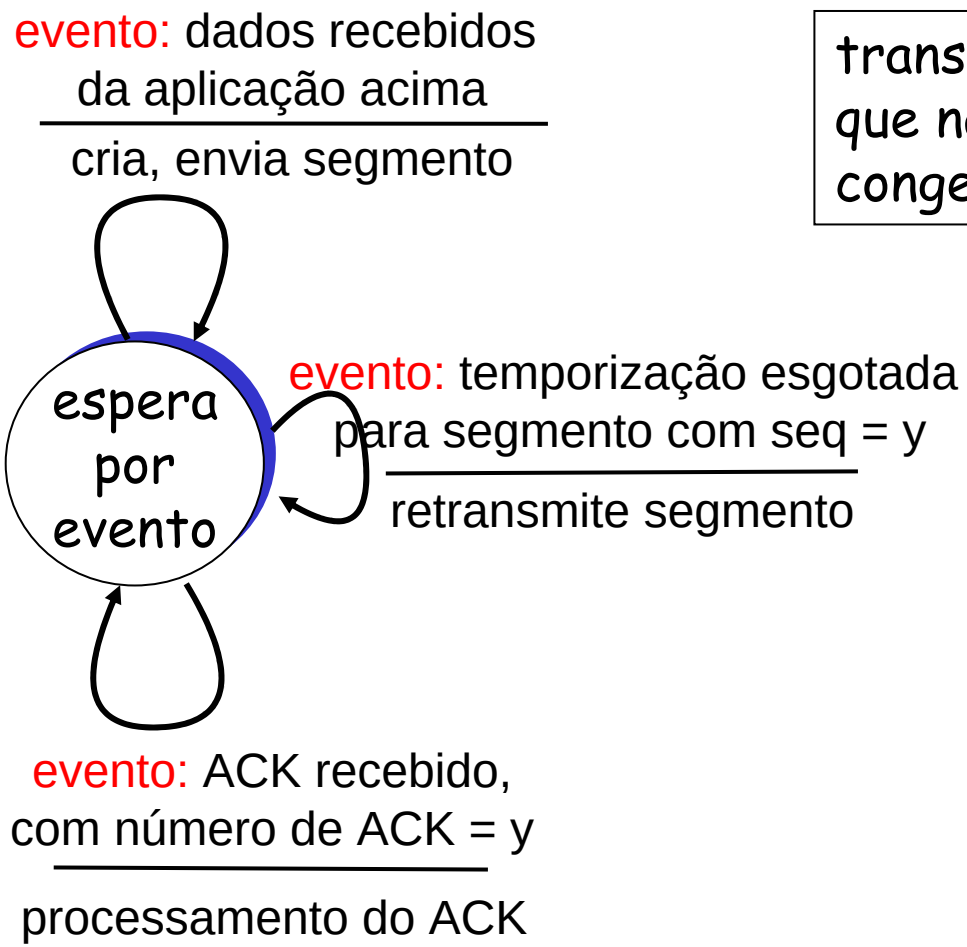
- número do primeiro byte no segmentos de dados

ACKs:

- número do próximo byte esperado do outro lado
- ACK cumulativo Q: como o receptor trata segmentos foram de ordem
- A especificação do TCP não define, fica a critério do implementador



TCP: transferência de dados confiável



transmissor simplificado, assumindo que não há controle de fluxo nem de congestionamento

TCP: transferência confiável

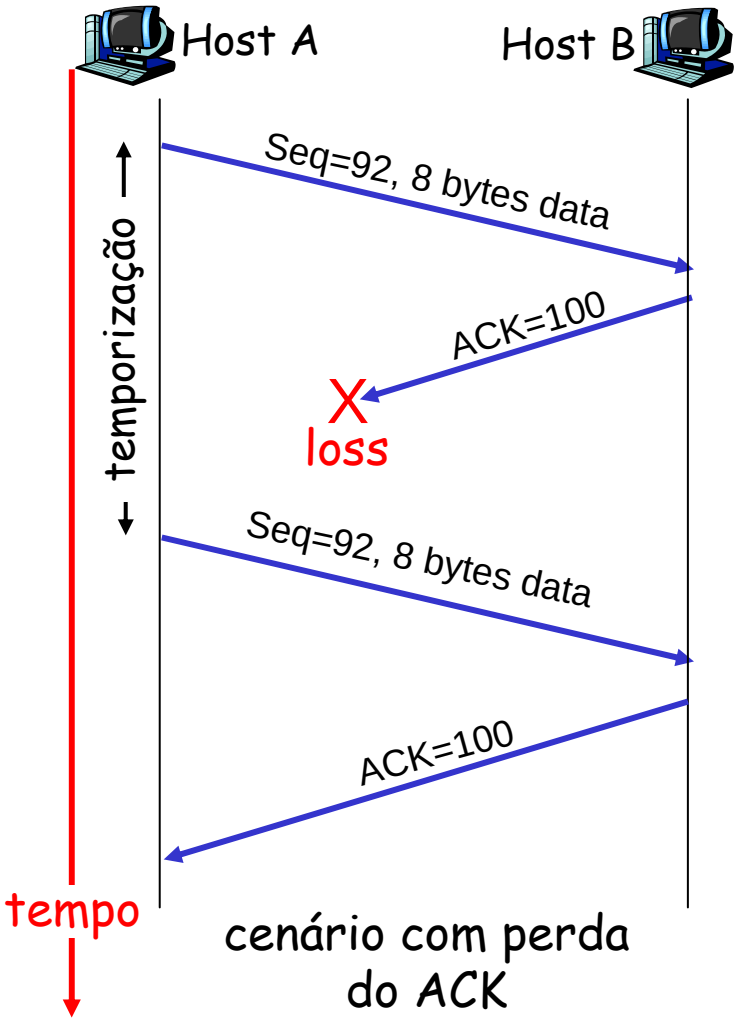
Transmissor
TCP simplificado

```
00  sendbase = initial_sequence number
01   nextseqnum = initial_sequence number
02
03    loop (forever) {
04      switch(event)
05      event: dados recebidos da aplicação acima
06           cria segmento TCP com número de seqüência nextseqnum
07           inicia temporizador para segmento nextseqnum
08           passa segmento ao IP
09           nextseqnum = nextseqnum + length(data)
10       event: esgotamento de temp. para segmento com seq =  y
11           retransmite segmento com seq = y
12           calcule nova temporização para o segmento y
13           reinicia temporizador para número de seqüência  y
14       event: ACK recebido, com valor y no campo de ACK
15           if (y > sendbase) { /* ACK cumulativo de todos os dados até y */
16               cancela todas as temporizações para segmentos com seq < y
17                sendbase = y
18                }
19           else { /* um ACK duplicata para um segmento já reconhecido */
20                incrementa numero de duplicatas de ACKs recebidas para y
21                if (numero de duplicatas de  ACKS recebidas para y == 3) {
22                   /* TCP fast retransmit */
23                  reenvia segmento com número de seqüência  y
24                  reinicia temporização para segmento y
25               }
26      }  /* end of loop forever */
```

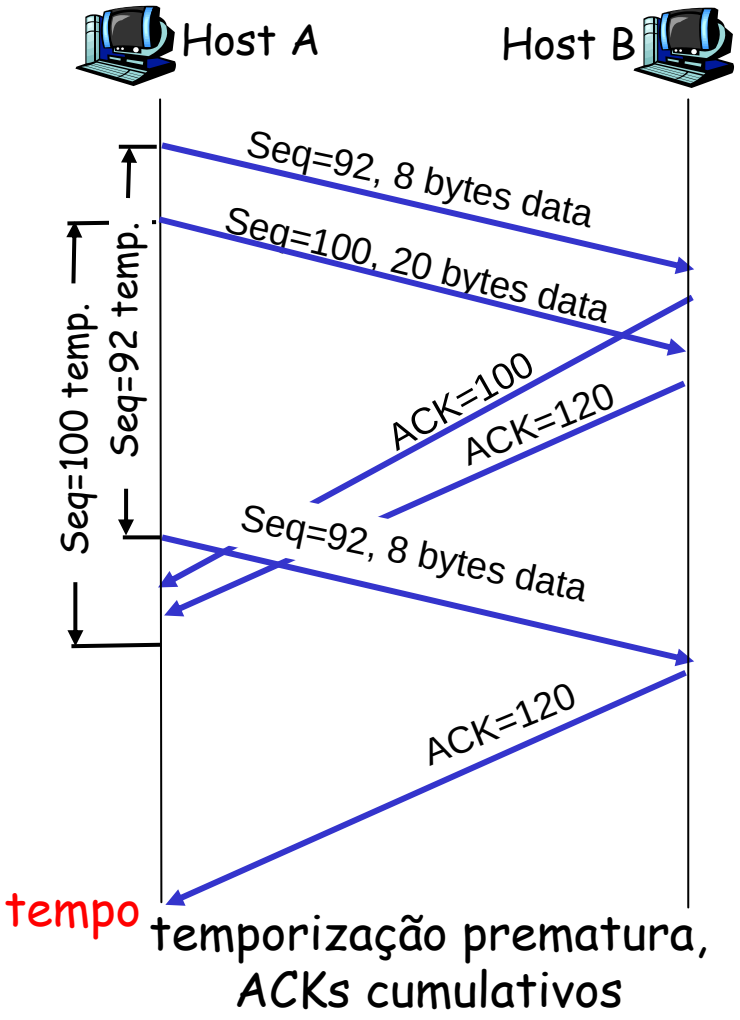
Geração de ACK [RFC 1122, RFC 2581]

Evento	Ação do TCP Receptor
segmento chega em ordem, não há lacunas, segmentos anteriores já aceitos	ACK retardado. Espera até 500ms pelo próximo segmento. Se não chegar, envia ACK
segmento chega em ordem, não há lacunas, um ACK atrasado pendente	imediatamente envia um ACK cumulativo
segmento chega fora de ordem número de seqüência chegou maior: gap detetado	envia ACK duplicado, indicando número de seqüência do próximo byte esperado
chegada de segmento que parcial ou completamente preenche o gap	reconhece imediatamente se o segmento começa na borda inferior do gap

TCP: cenários de retransmissão



cenário com perda do ACK



temporização prematura, ACKs cumulativos

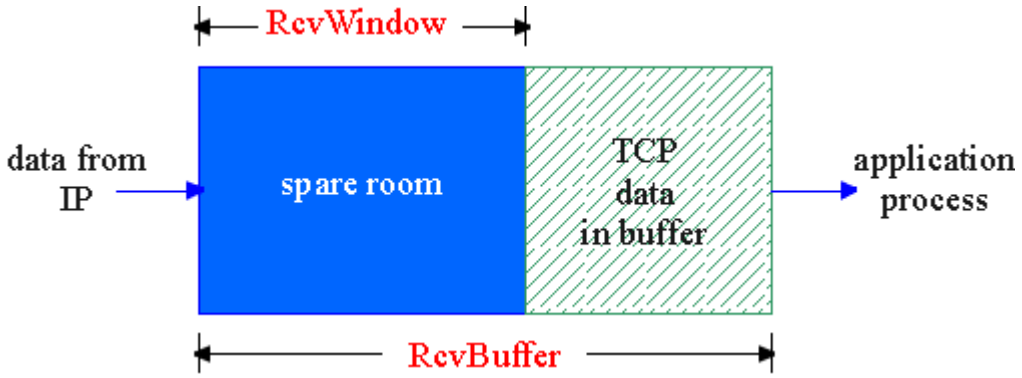
TCP: Controle de Fluxo

controle de fluxo

transmissor não deve esgotar os buffers de receção enviando dados rápido demais

`RcvBuffer` = tamanho do Buffer de recepção do TCP

`RcvWindow` = total de espaço livre no buffer



armazenamento de recepção

receptor: explicitamente informa o transmissor da área livre no buffer (dinamicamente mudando)

- **`campo RcvWindow`** no segmento TCP

transmissor: mantém a quantidade de dados transmitidos mas não reconhecidos menor que o último **`RcvWindow`** recebido

TCP Round Trip Time e Temporização

Q: como escolher o valor da temporização do TCP?

- maior que o RTT
  - nota: RTT varia
- muito curto: temporização prematura
  - retransmissões desnecessárias
- muito longo: a reação à perda de segmento fica lenta

Q: Como estimar o RTT?

- **`SampleRTT`**: tempo medido da transmissão de um segmento até a respectiva confirmação
  - ignora retransmissões e segmentos reconhecidos de forma cumulativa
- **`SampleRTT`** varia de forma rápida, é desejável um amortecedor para a estimativa do RTT
  - usar várias medidas recentes, não apenas o último **`SampleRTT`** obtido

TCP Round Trip Time e Temporização

`EstimatedRTT = (1-x)*EstimatedRTT + x*SampleRTT`

- Média móvel com peso exponential
- influencia de uma dada amostra decresce de forma exponencial
- valor típico de x: 0.1

Definindo a temporização

- **`EstimtedRTT`** mais “margem de segurança”
- grandes variações no **`EstimatedRTT ->`** maior margem de segurança

```
Temporização = EstimatedRTT + 4*Desvios
Desvios = (1-x)*Desvio +
          x*|SampleRTT-EstimatedRTT|
```

TCP Estabelecimento de Conexão

TCP transmissor estabelece conexão com o receptor antes de trocar segmentos de dados

- initializar variáveis:
  - números de seqüência
  - buffers, controle de fluxo (ex. **`RcvWindow`**)
- *cliente:* iniciador da conexão

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

- *servidor:* chamado pelo cliente

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

Three way handshake:

Passo 1: sistema final cliente envia TCP SYN ao servidor
  - especifica número de seqüência inicial

Passo 2: sistema final servidor que recebe o SYN, responde com segmento SYNACK
  - reconhece o SYN recebido
  - aloca buffers
  - especifica o número de seqüência inicial do servidor

Passo 3: o sistema final cliente reconhece o SYNACK

TCP Término de Conexão

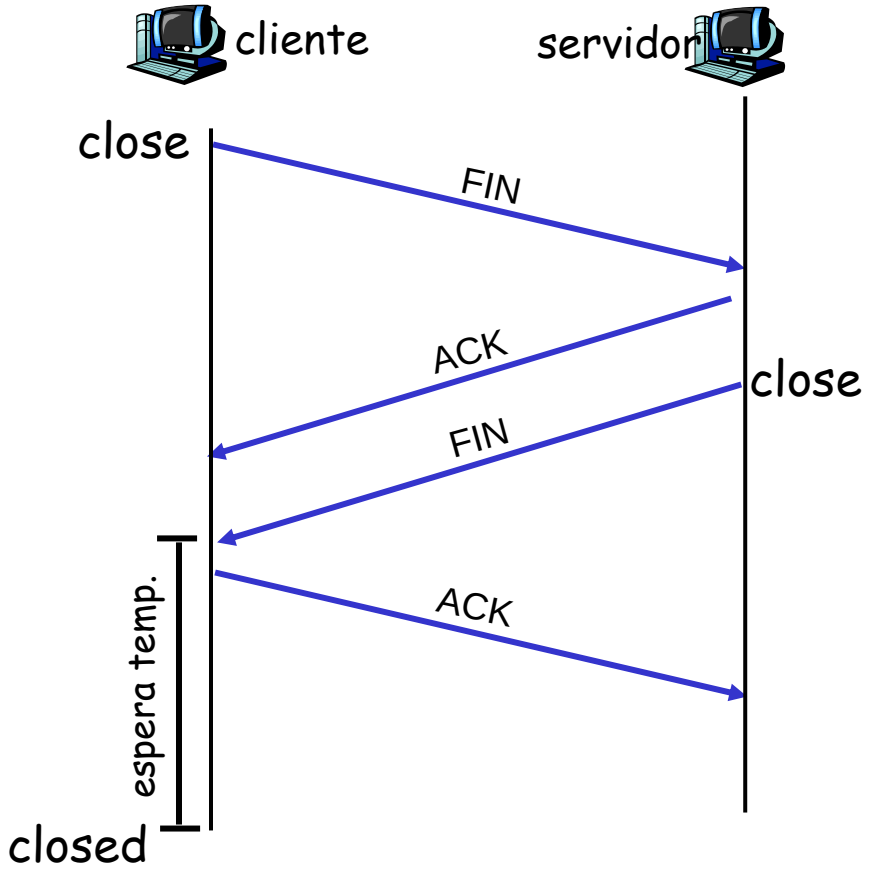
Fechando uma conexão:

cliente fecha o socket:

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

Passo 1: o cliente envia o segmento TCP FIN ao servidor

Passo 2: servidor recebe FIN, responde com ACK. Fecha a conexão, envia FIN.

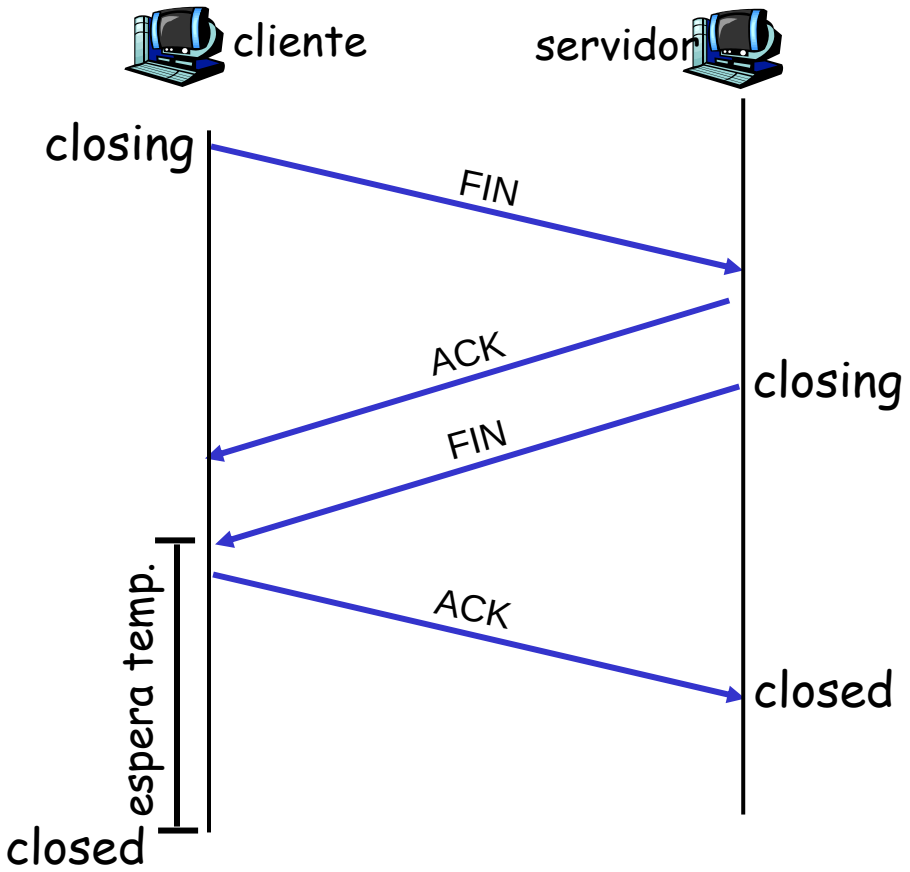


TCP Término de Conexão

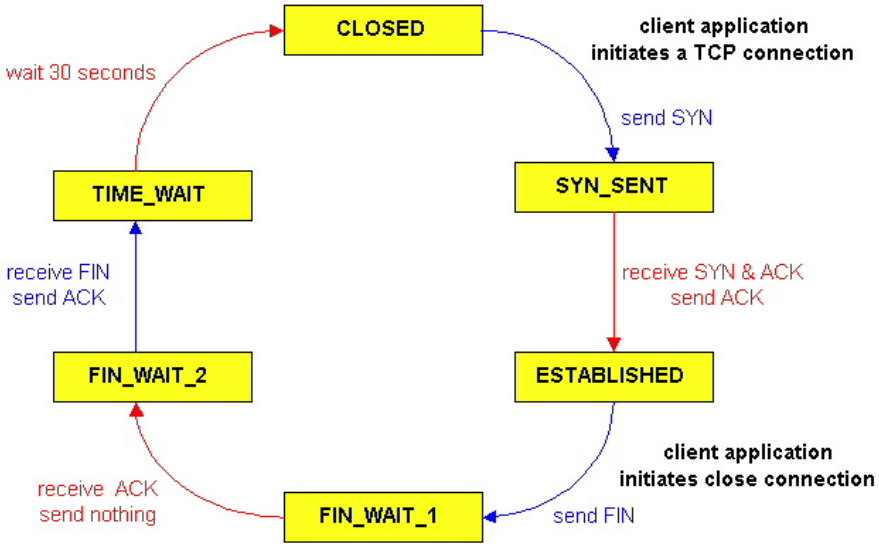
Passo 3: cliente recebe FIN, responde com ACK.

- Entra “espera temporizada” - vai responder com ACK a FINs recebidos

Passo 4: servidor, recebe ACK. Conexão fechada.

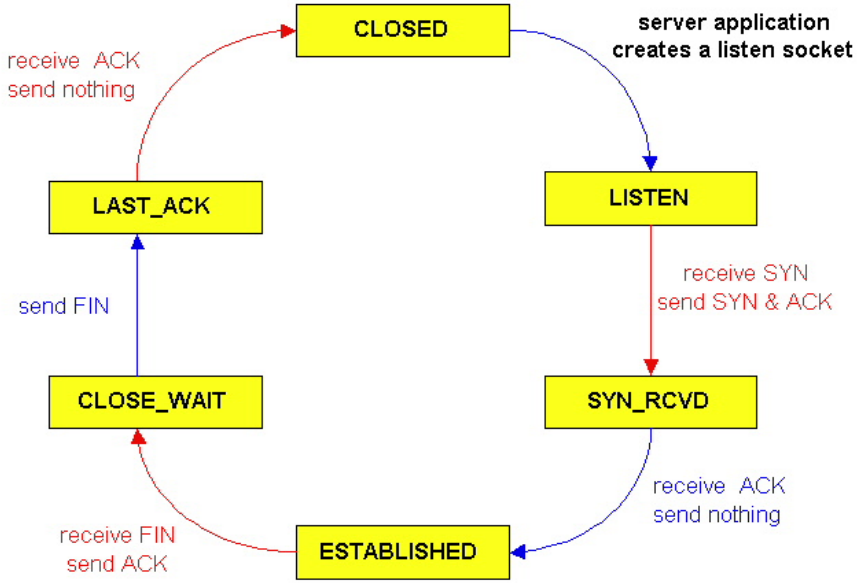


TCP Controle de Conexão



Estados do Cliente

Estados do Servidor



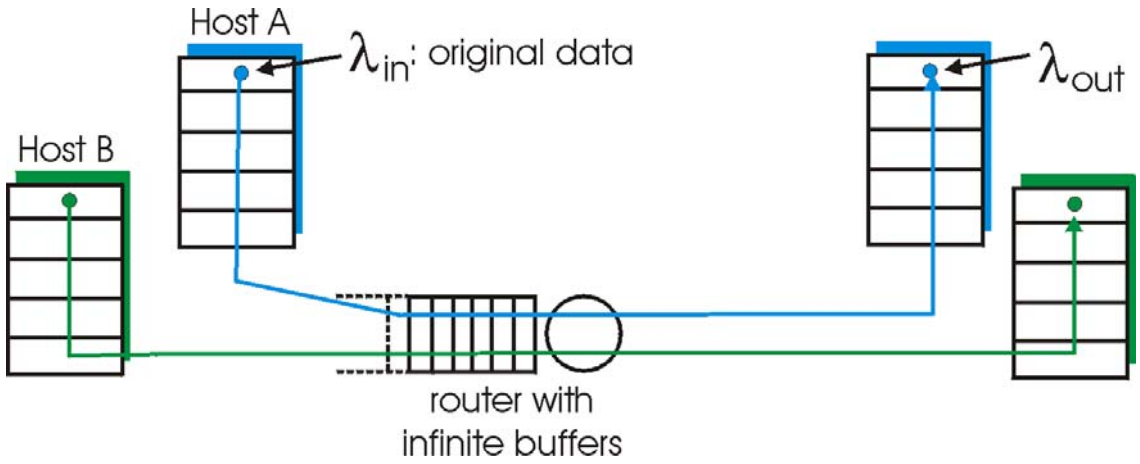
Princípios de Controle de Congestionamento

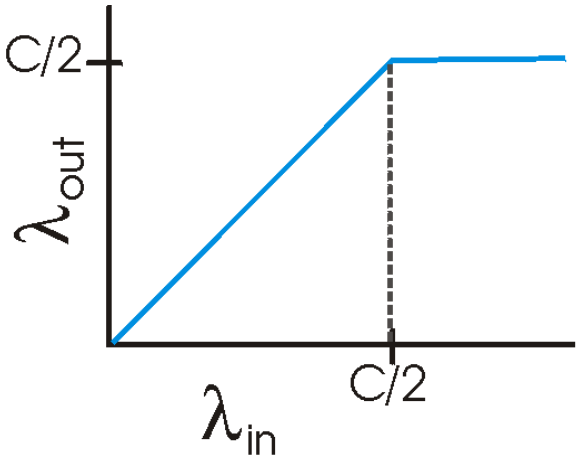
Congestionamento:

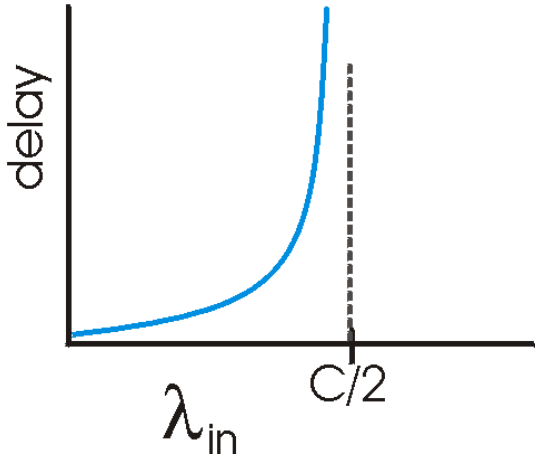
- informalmente: “muitas fontes enviando dados acima da capacidade da *rede* tratá-los”
- diferente de controle de fluxo!
- sintomas:
  - perda de pacotes (saturação de buffer nos roteadores)
  - atrasos grandes (filas nos buffers dos roteadores)
- um dos 10 problemas mais importantes na Internet!

Causas/custos do congestionamento: cenário 1

- dois transmissores, dois receptores
- um roteador, buffers infinitos
- não há retransmissão



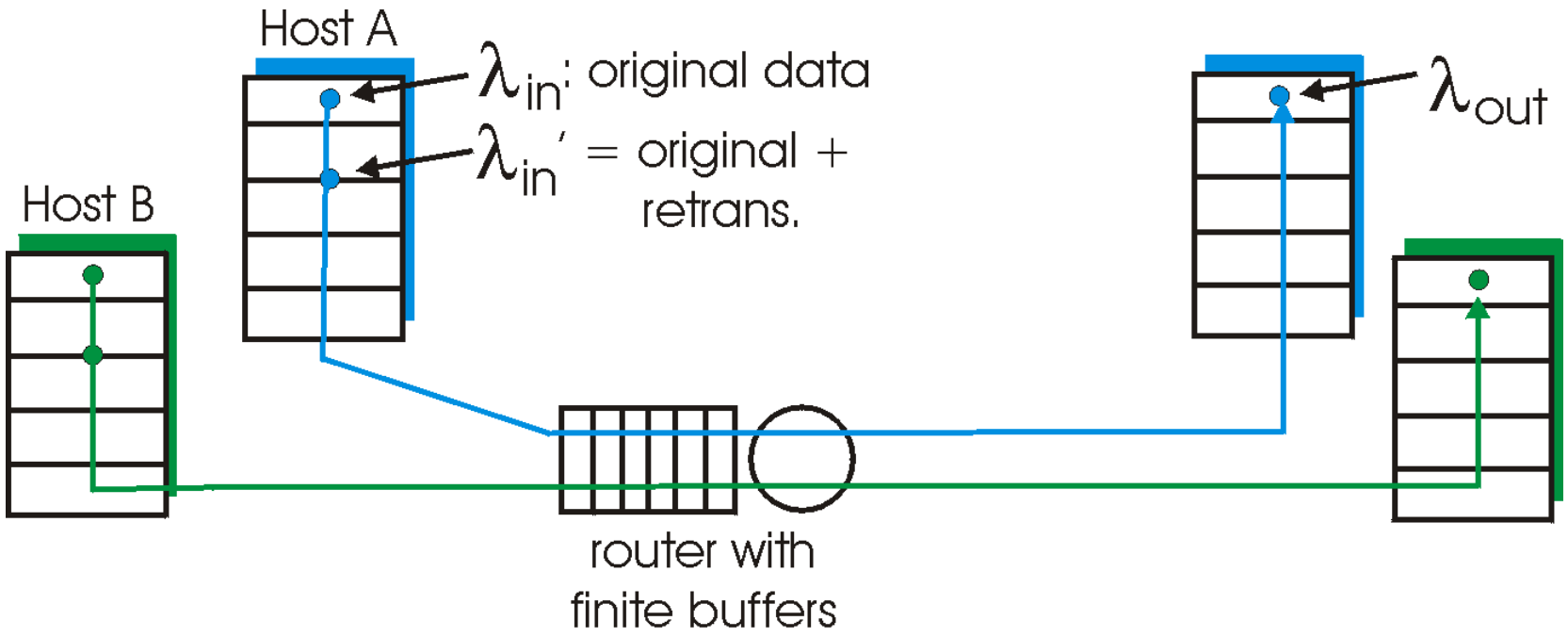




- grandes atrasos quando congestionado
- máxima vazão obtenível

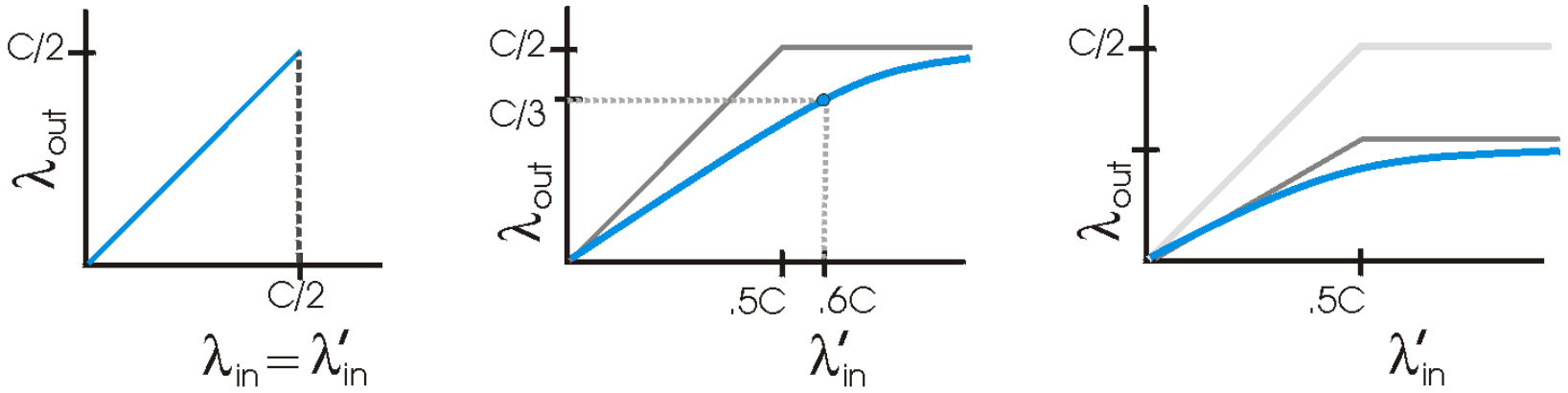
Causas/custos do congestionamento: cenário 2

- um roteador, buffers *finitos*
- transmissor reenvia pacotes perdidos



Causas/custos do congestionamento: cenário 2

- sempre vale : $\lambda_{in} = \lambda_{out}$ (tráfego bom)
- "perfeita" retransmissão somente quando há perdas: $\lambda'_{in} > \lambda_{out}$
- retransmissão de pacotes atrasados (não perdidos) torna λ'_{in} maior (que o caso perfeito) para o mesmo λ_{out}



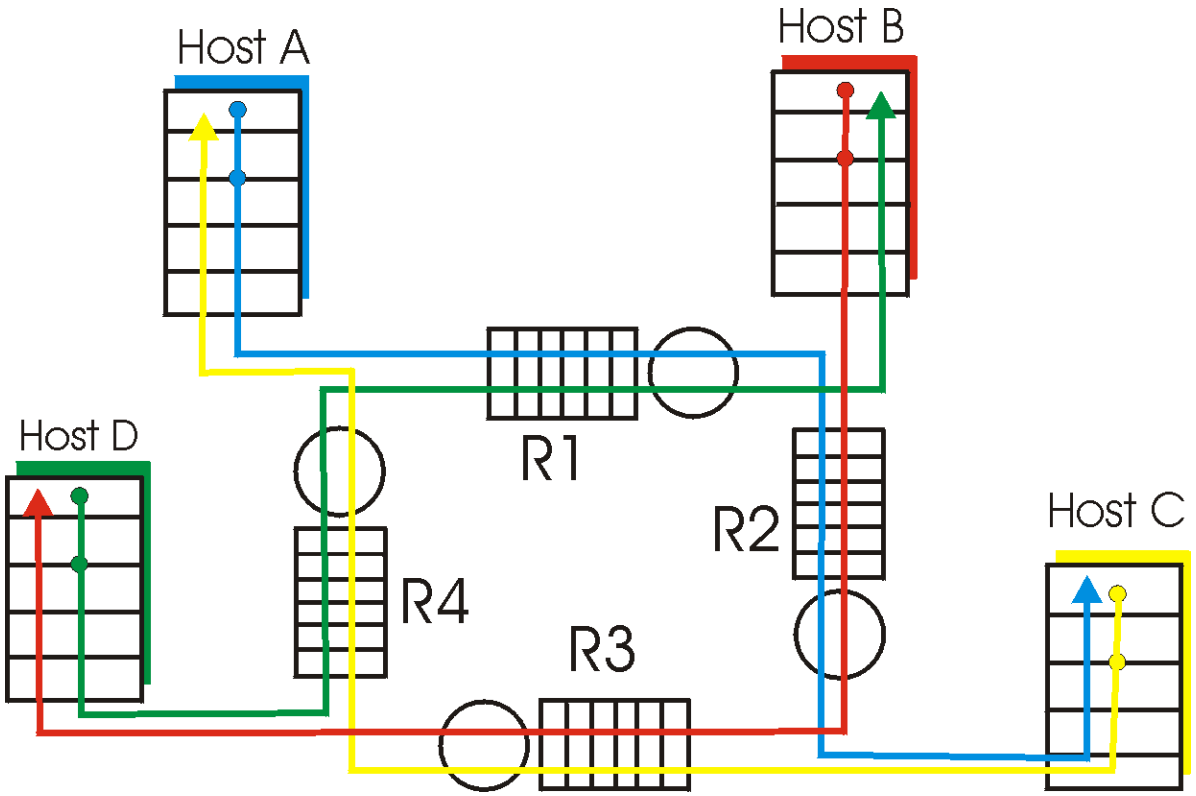
"custos" do congestionamento:

- mais trabalho (retransmissões) para um dado "tráfego bom"
- retransmissões desnecessárias: enlace transporta várias cópias do mesmo pacote

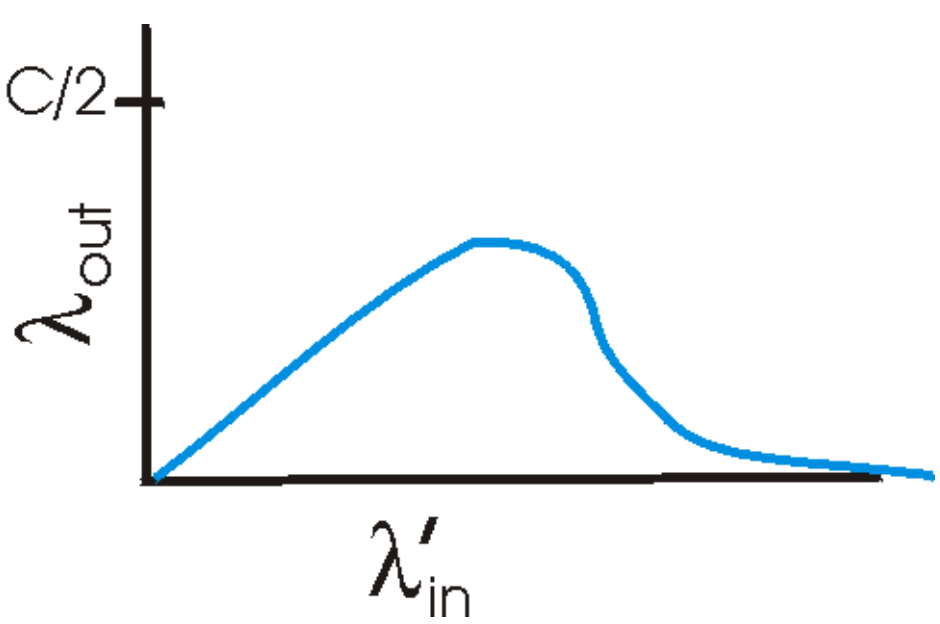
Causas/custos do congestionamento: cenário 3

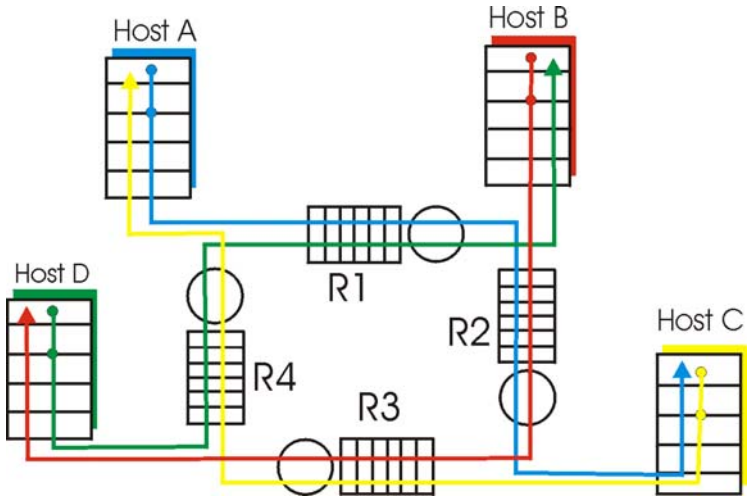
- quatro transmissores
- caminhos com múltiplos saltos
- temporizações/retransmissões

Q: o que acontece quando λ_{in} e λ'_{in} aumentam ?



Causas/custos do congestionamento: cenário 3





Outro “custo” do congestionamento:

- quando pacote é descartado, qualquer capacidade de transmissão que tenha sido anteriormente usada para aquele pacote é desperdiçada!

Abordagens do problema de controle de congestionamento

Existem duas abordagens gerais para o problema de controle de congestionamento:

Controle de congestionamento fim-a-fim:

- não usa realimentação explícita da rede
- congestionamento é inferido a partir das perdas e dos atrasos observados nos sistemas finais
- abordagem usada pelo TCP

Controle de congestionamento assistido pela rede:

- roteadores enviam informações para os sistemas finais
  - bit único indicando o congestionamento (SNA, DECbit, TCP/IP ECN, ATM)
  - taxa explícita do transmissor poderia ser enviada

Estudo de caso: controle de congestionamento do serviço ATM ABR

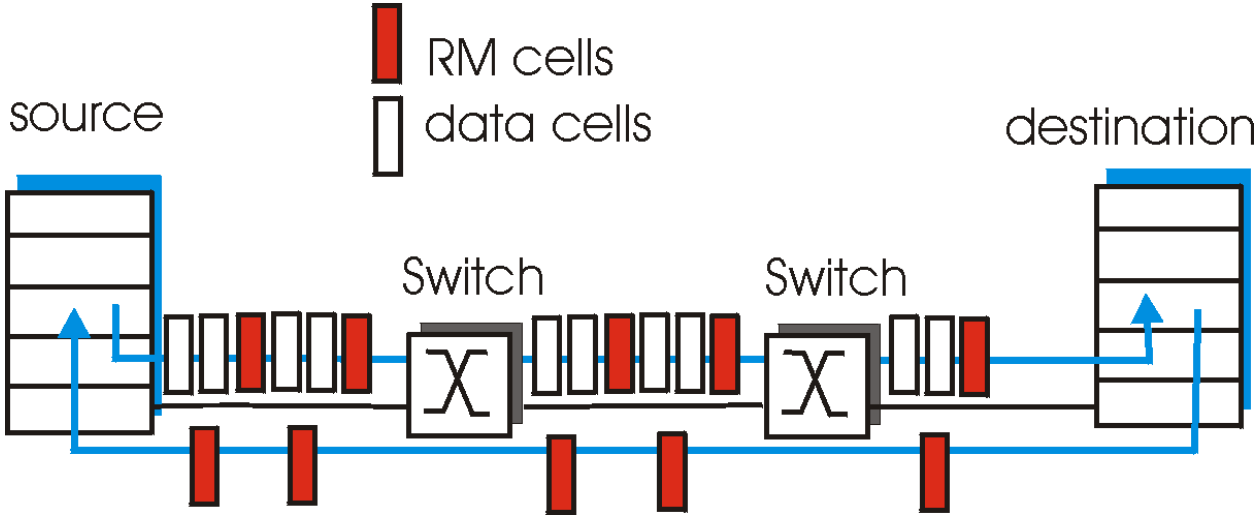
ABR: available bit rate:

- “serviço elástico”
- se o caminho do transmissor está pouco usado:
  - transmissor pode usar a banda disponível
- se o caminho do transmissor está congestionado:
  - transmissor é limitado a uma taxa mínima garantida

células RM (resource management) :

- enviadas pelo transmissor, entremeadas com as células de dados
- bits nas células RM são usados pelos comutadores (“*assistida pela rede*”)
  - NI bit: não aumentar a taxa (congestionamento leve)
  - CI bit: indicação de congestionamento
- as células RM são devolvidos ao transmissor pelo receptor, com os bits de indicaçaõ intactos

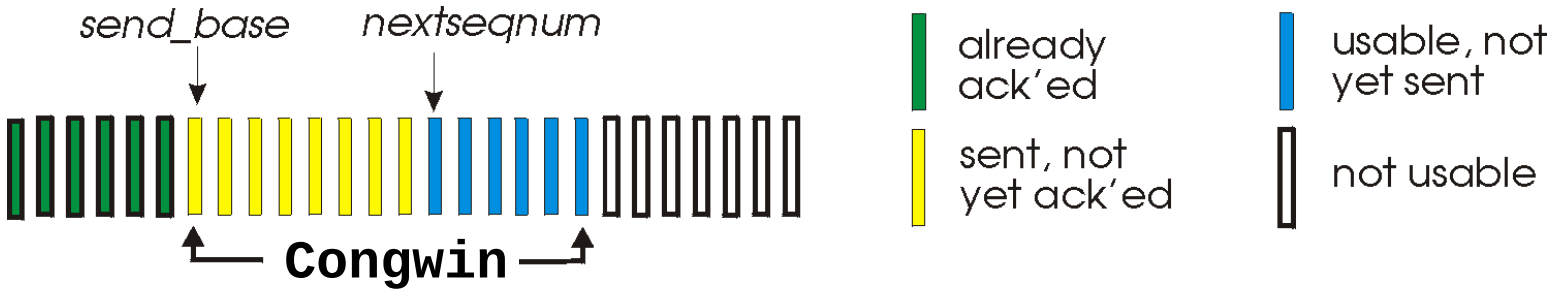
Estudo de caso: controle de congestionamento do serviço ATM ABR



- campo ER (explicit rate) de dois bytes nas células RM
  - comutador congestionado pode reduzir o valor de ER nas células
  - o transmissor envia dados de acordo com esta vazão mínima suportada no caminho
- bit EFCI nas células de dados: marcado como 1 pelos comutadores congestionados
  - se a célula de dadosque precede a célula RM tem o bit EFCI com valor 1, o receptor marca o bit CI na célula RM devolvida

TCP: Controle Congestionamento

- controle fim-a-fim (não há assistência da rede)
- taxa de transmissão é limitada pelo tamanho da janela, **`Congwin`**, sobre os segmentos:



- w segmentos, cada um com MSS bytes enviados em um RTT:

$$\text{vazão} = \frac{w * MSS}{RTT} \text{ Bytes/seg}$$

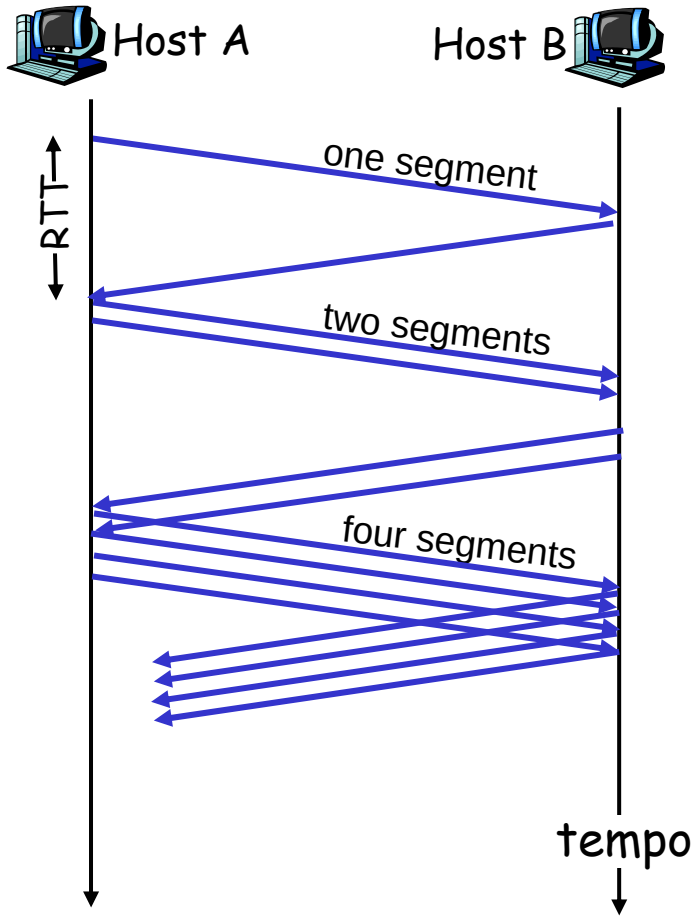
TCP: Controle Congestionamento

- "teste" para reconhecer a taxa possível:
  - idealmente: transmitir tão rápido quanto possível (**`Congwin`** tão grande quanto possível) sem perdas
  - *aumentar* **`Congwin`** até que ocorra perda (congestionamento)
  - perda: *diminuir* **`Congwin`**, então ir testando (aumentando) outra vez
- duas "fases""
  - slow start
  - congestion avoidance
- variáveis importantes:
  - **`Congwin`**
  - **`threshold:`** define o limite entre a fase slow start e a fase congestion avoidance

TCP Slowstart

algoritmo Slowstart

initializar: Congwin = 1
para (cada segmento reconhecido
    Congwin++
até (evento perda OU
    CongWin > threshold)



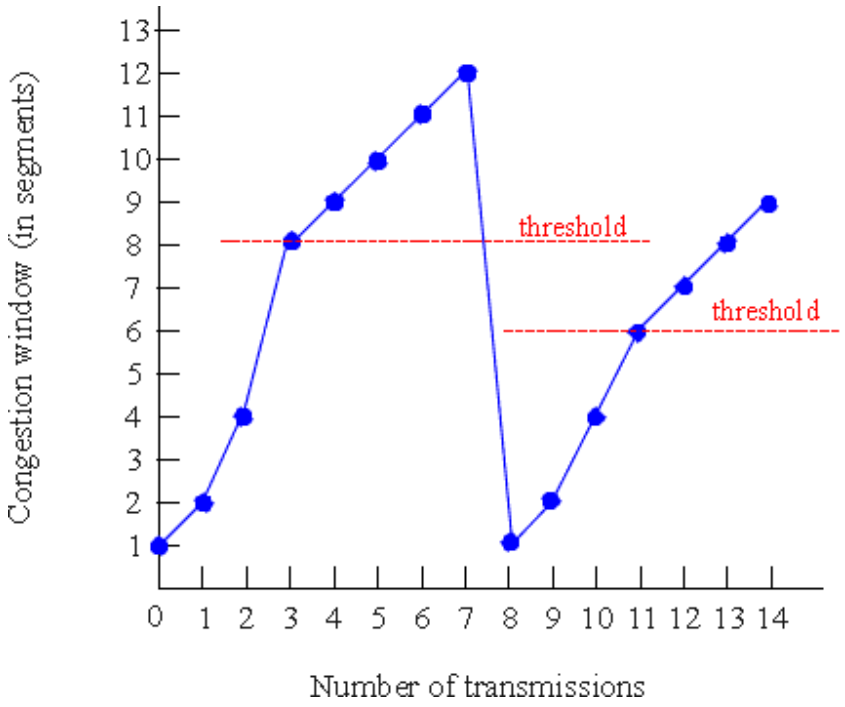
- aumento exponencial (por RTT) no tamanho da janela (não tão lento!)
- evento de perda : temporização (Tahoe TCP) e/ou 3 ACKs duplicados (Reno TCP)

TCP: Congestion Avoidance

Congestion avoidance

```
/* acabou slowstart        */
/* Congwin > threshold */
Até (evento perda) {
  cada w segmentos reconhecidos:
      Congwin++
  }
threshold = Congwin/2
Congwin = 1
realiza slowstart
```

1



1: TCP Reno pula a fase slowstart (recuperaçaõ rápida) após três ACKs duplicados

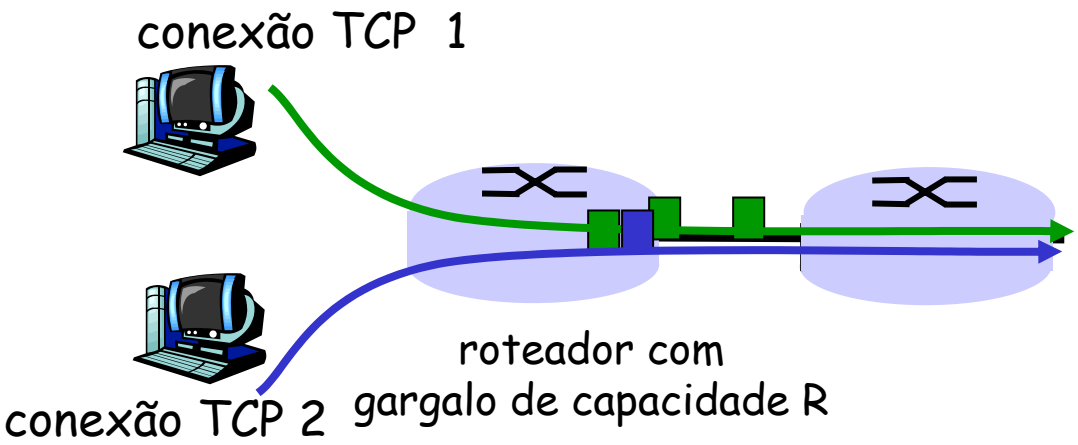
AIMD

TCP congestion avoidance:

- AIMD: *aumento aditivo, redução multiplicativa*
  - aumenta a janela de 1 a cada RTT
  - diminui a janela por um fator de 2 em caso de evento perda

TCP Eqüidade

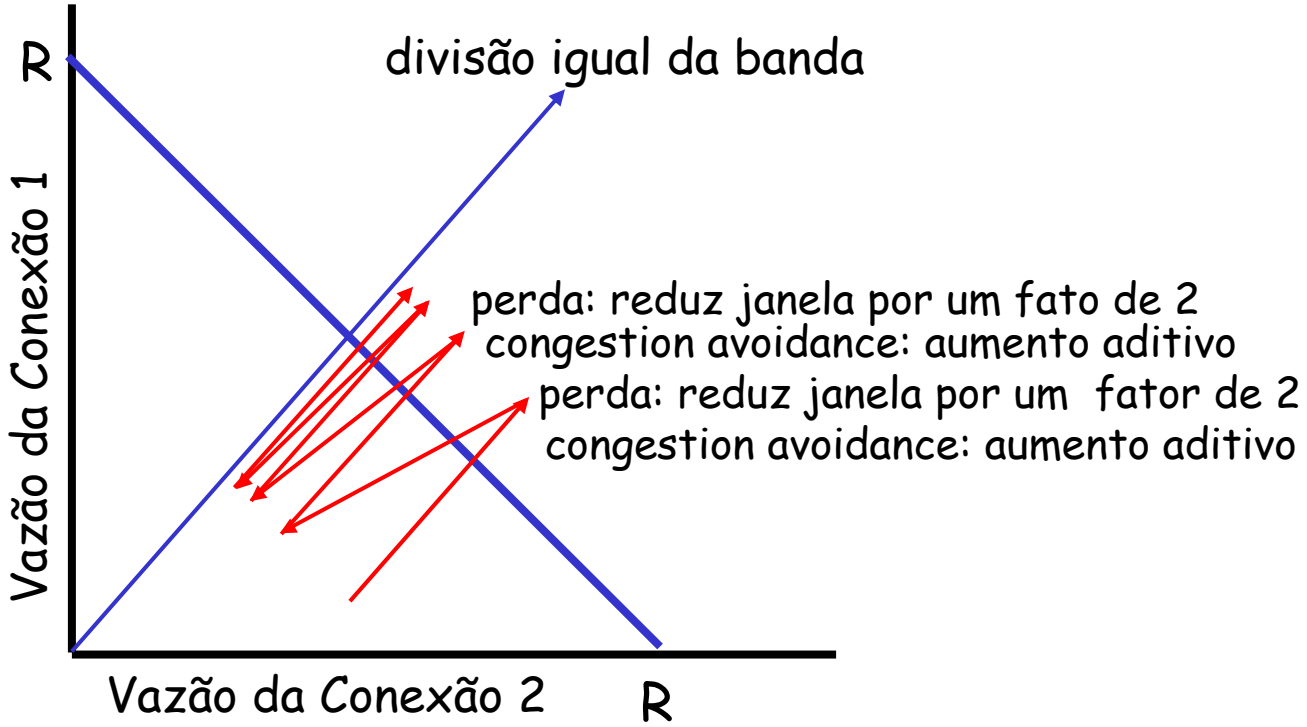
Objetivo: se N sessões TCP devem passar pelo mesmo gargalo, cada uma deve obter 1/N da capacidade do enlace



Porque o TCP é justo?

Duas sessões competindo pela banda:

- O aumento aditivo fornece uma inclinação de 1, quando a vazão aumenta
- redução multiplicativa diminui a vazão proporcionalmente



TCP: modelagem da latência

Q: Quanto tempo demora para receber um objeto de um servidor Web após enviar um pedido?

- estabelecimento de conexão TCP
- atraso de transferência de dados

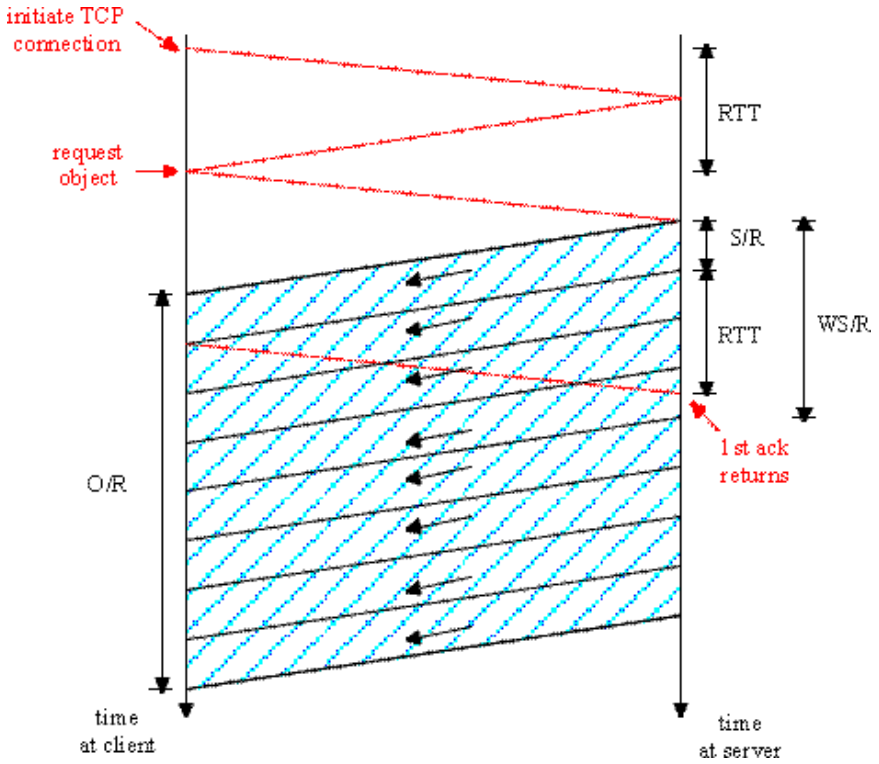
Notação, hipóteses:

- Assuma um enlace entre o cliente e o servidor com taxa de dados R
- Assuma: janela de congestionamento fixa, W segmentos
- S: MSS (bits)
- O: tamanho do objeto (bits)
- não há retransmissões (sem perdas e corrupção de dados)

Dois casos a considerar:

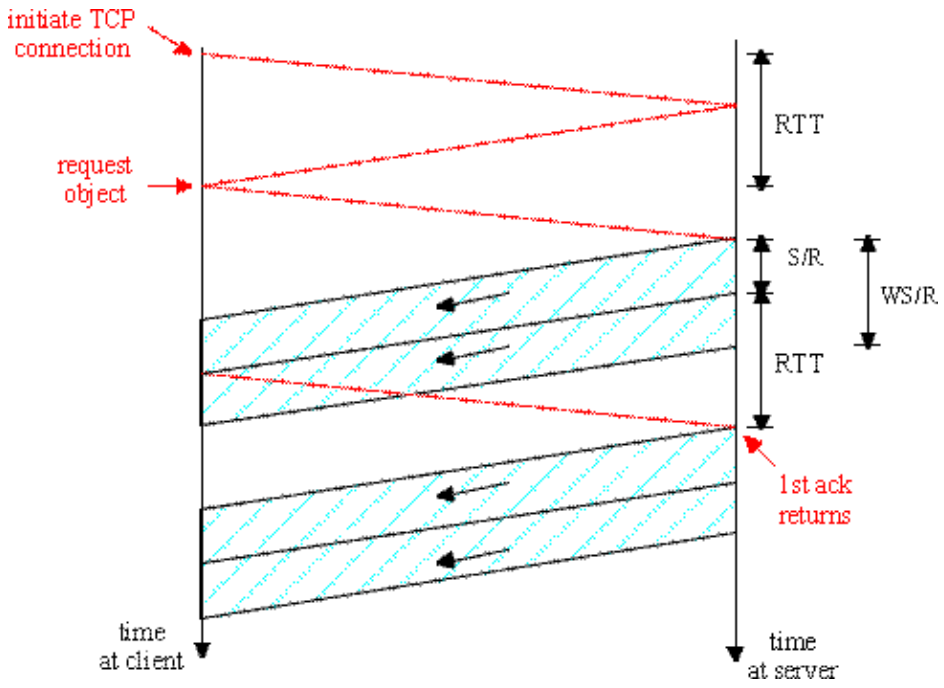
- WS/R > RTT + S/R: ACK para o primeiro segmento retorna antes de se esgotar a janela de transmissão de dados
- WS/R < RTT + S/R: espera pelo depois de esgotar a janela de transmissão de dados

TCP: modelagem da latência



Caso 1: latencia = 2RTT + O/R

K:= O/WS



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

TCP Modelagem de Latência: Slow Start

- Agora suponha que a janela cresce de acordo com os procedimentos da fase slow start.
- Vamos mostrar que a latência de um objeto de tamanho O é:

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

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

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

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

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

TCP Modelagem de Latência: Slow Start (cont.)

Exemplo:

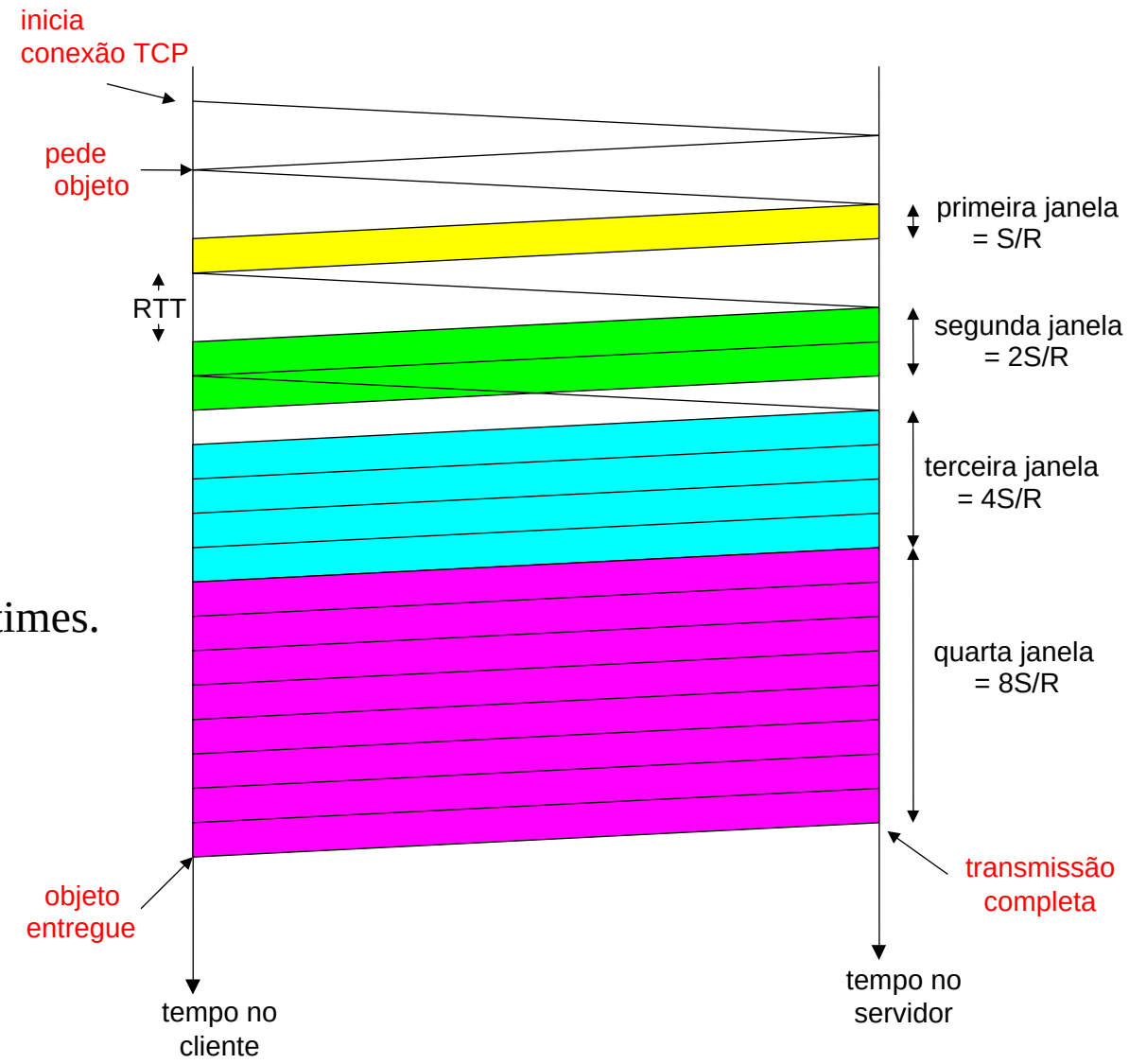
O/S = 15 segmentos

K = 4 janelas

Q = 2

P = min{K-1,Q} = 2

Servidor bloqueado P=2 times.



TCP Modelagem de Latência: Slow Start (cont.)

$\frac{S}{R} + RTT$ = tempo quando o servidor inicia o envio do segmento até quando o servidor recebe reconhecimento

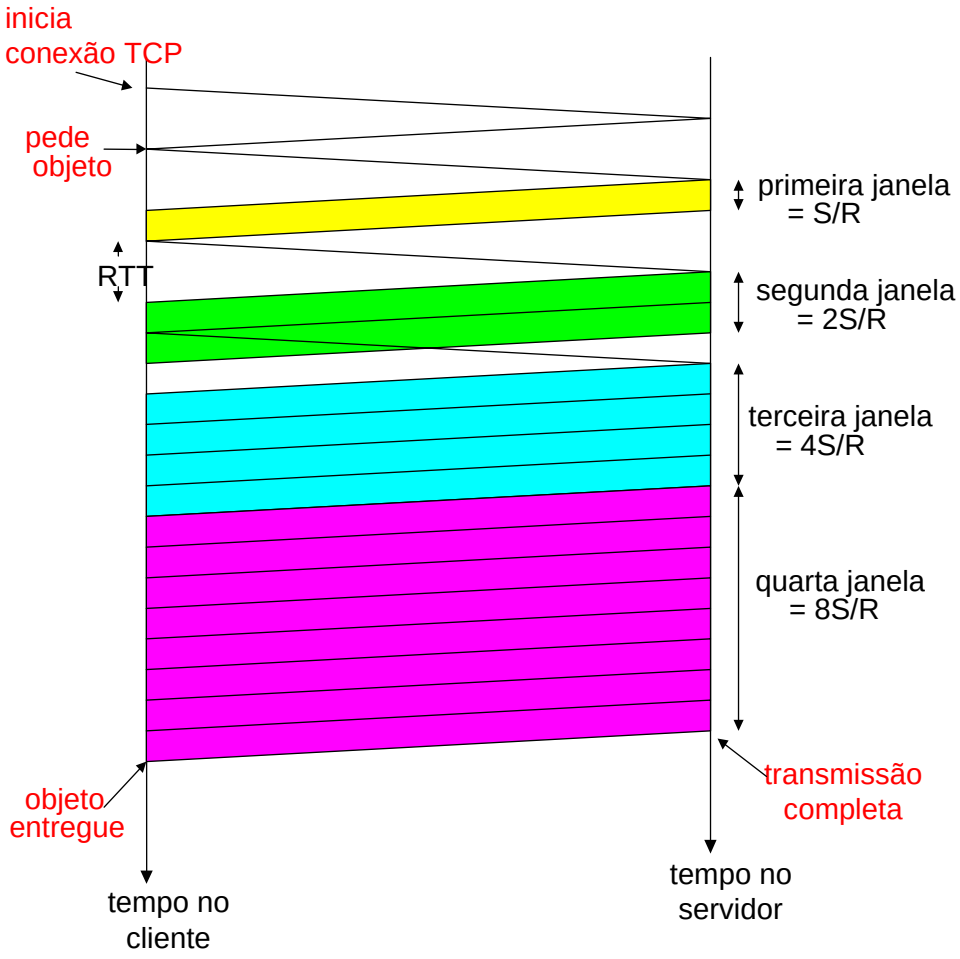
$2^{k-1}\frac{S}{R}$ = tempo para enviar a k-ésima janela

$\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^{+}$ = tempo de bloqueio após a k-ésima janela

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

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

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



Capítulo 3: Resumo

- princípios por trás dos serviços da camada de transporte:
  - multiplexação/demultiplexação
  - transferência de dados confiável
  - controle de fluxo
  - controle de congestionamento
- instantiação e implementação na
  - UDP
  - TCP

A seguir:

- saímos da “borda” da rede (camadas de aplicação e de transporte)
- vamos para o “núcleo” da rede