3 TCP Round Trip Time e temporização

P.: 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

P.: Como estimar o RTT?

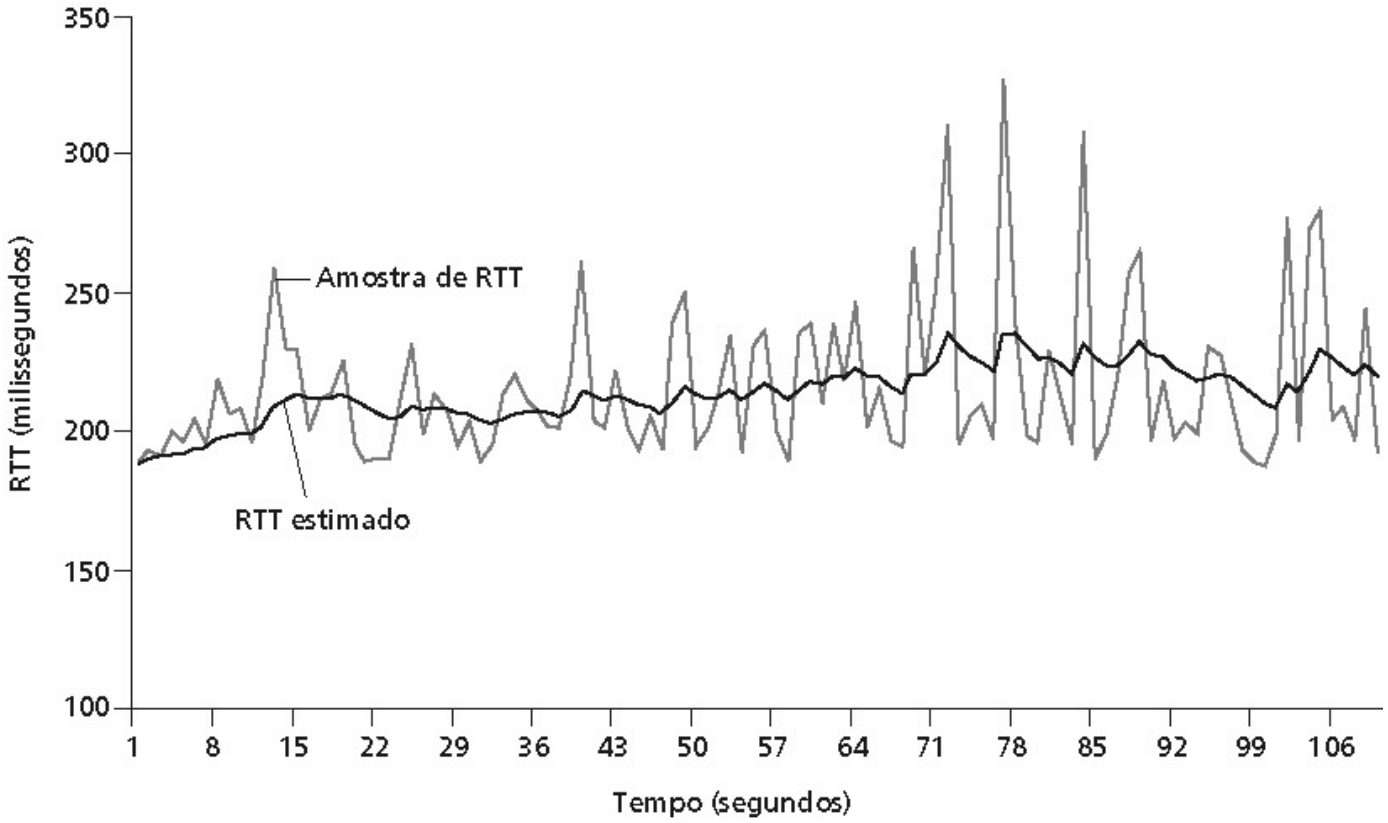
- SampleRTT: tempo medido da transmissão de um segmento até a respectiva confirmação
  - Ignora retransmissões e segmentos reconhecidos de forma cumulativa
- **SampleRTT** varia de forma rápida, é desejável um amortecedor para a estimativa do RTT
  - Usar várias medidas recentes, não apenas o último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

3 TCP Round Trip Time e temporização

$$\mathbf{EstimatedRTT = (1-\alpha)*EstimatedRTT + \alpha*SampleRTT}$$

- Média móvel com peso exponencial
- Influência de uma dada amostra decresce de forma exponencial
- Valor típico: α = 0,125

3 Exemplos de estimativa do RTT



3 TCP Round Trip Time e temporização

Definindo a temporização

- **`EstimatedRTT`** mais "margem de segurança"
  - Grandes variações no **`EstimatedRTT ->`** maior margem de segurança
- Primeiro estimar o quanto o SampleRTT se desvia do EstimatedRTT:

$$\texttt{DevRTT} = (1-\beta)*\texttt{DevRTT} + \beta*|\texttt{SampleRTT}-\texttt{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

Então ajustar o intervalo de temporização

$$\texttt{TimeoutInterval} = \texttt{EstimatedRTT} + 4*\texttt{DevRTT}$$

3 Camada de transporte

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

3 TCP: transferência de dados confiável

- TCP cria serviços de rdt em cima do serviço não-confiável do IP
- Pipelined segments
- ACKs cumulativos
- TCP usa tempo de retransmissão simples
- Retransmissões são disparadas por:
  - Eventos de tempo de confirmação
  - ACKs duplicados
- Inicialmente, considere um transmissor TCP simplificado:
  - Ignore ACKs duplicados
  - Ignore controle de fluxo, controle de congestionamento

3 Eventos do transmissor TCP

Dado recebido da app:

- Crie um segmento com número de seqüência
- # seq é o número do byte-stream do 1º byte de dados no segmento
- Inicie o temporizador se ele ainda não estiver em execução (pense no temporizador para o mais antigo segmento não-confirmado)
- Tempo de expiração: `TimeOutInterval`

Tempo de confirmação:

- Retransmite o segmento que provocou o tempo de confirmação
- Reinicia o temporizador

ACK recebido:

- Quando houver o ACK de segmentos anteriormente não confirmados
  - Atualizar o que foi confirmado
  - Iniciar o temporizador se houver segmentos pendentes

3 Transmissor TCP (simplificado)

```
 NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
 loop (forever) {
   switch(event)
    event: dado recebido da aplicação acima
         cria segmento TCP com nº de seqüência NextSeqNum
         if (timer currently not running)
               start timer
         pass segment to IP
         NextSeqNum = NextSeqNum + length(data)
     event: tempo de confirmação do temporizador
         retransmit not-yet-acknowledged segment with
               smallest sequence number
         start timer
     event: ACK recebido, com valor do campo de ACK do y
         if (y > SendBase) {
               SendBase = y
              if (there are currently not-yet-acknowledged segments)
                     start timer
              }
 }  /* end of loop forever */
```

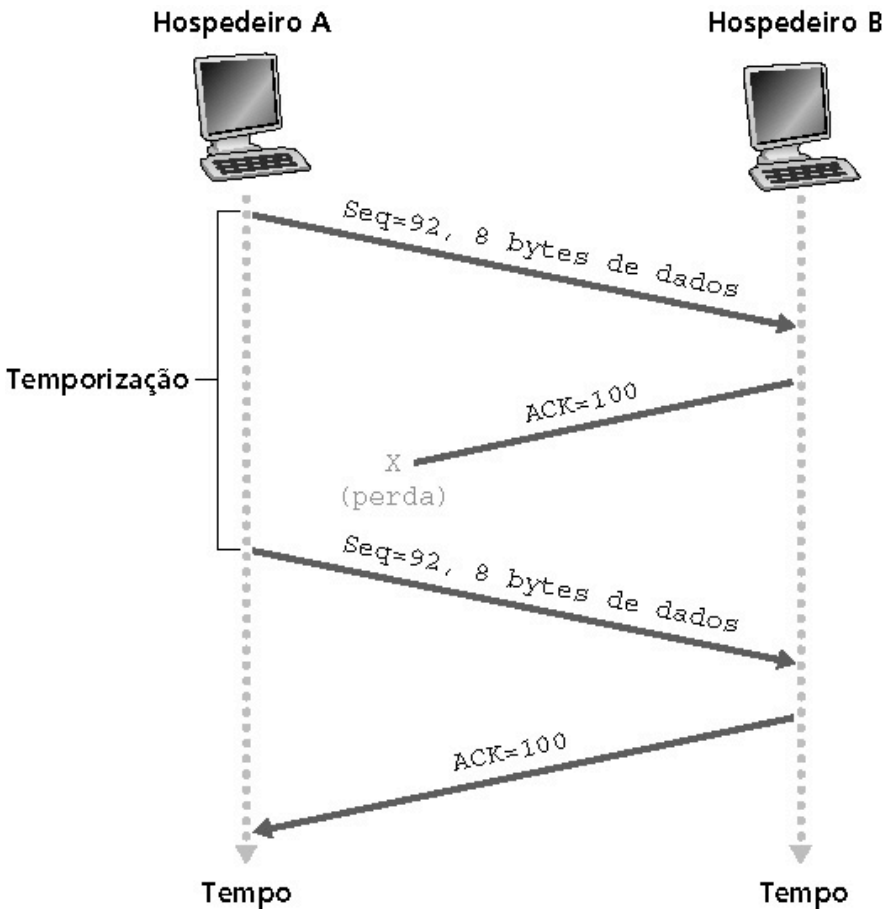
Comentário:

- SendBase-1
  último byte
  ACK
  cumulativo

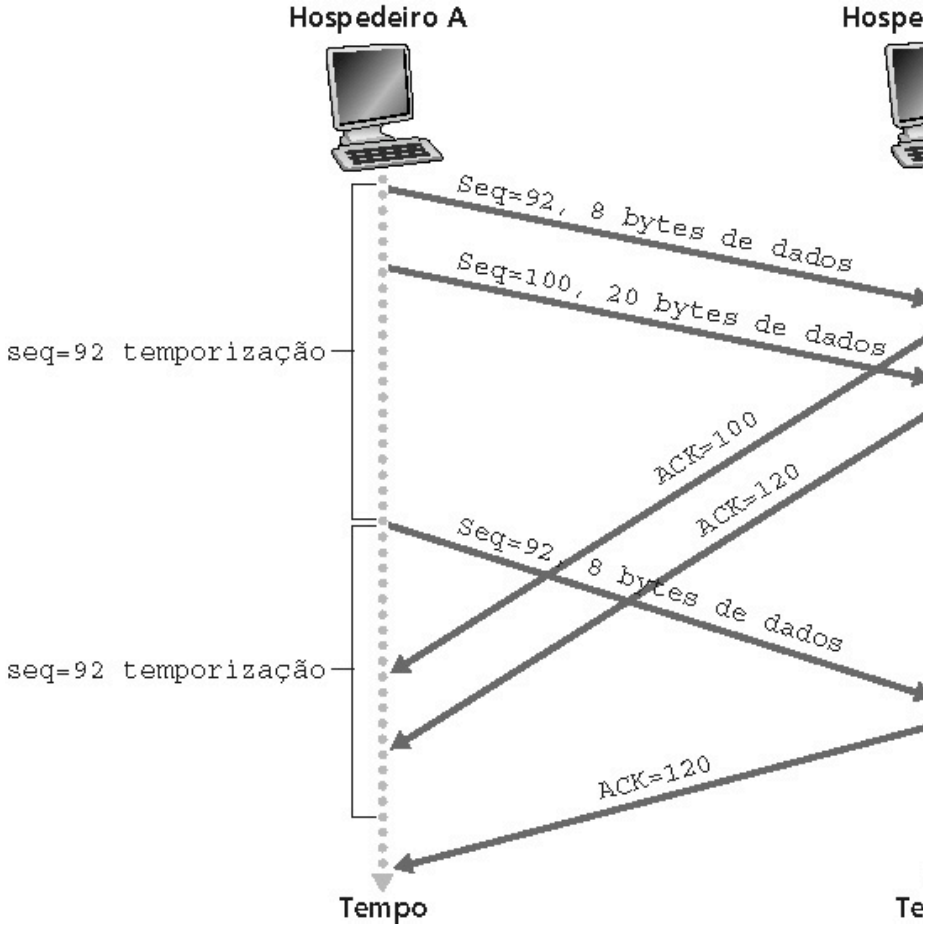
Exemplo:

- SendBase-1
  71; y= 73,
  então o
  receptor
  deseja
  73+ ; y >
  SendBase,
  então
  o novo dad
  confirmado

3 TCP: cenários de retransmissão

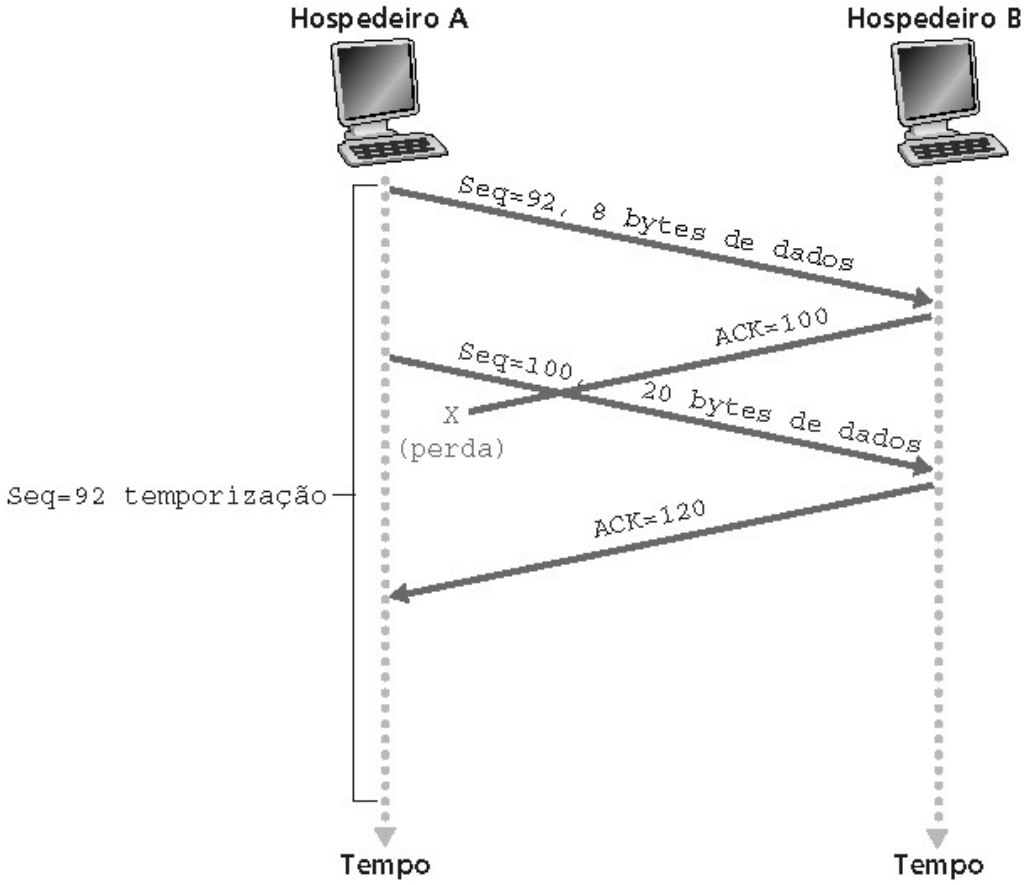


Cenário com perda do ACK



Temporização prematura, ACKs cumulativos

3 TCP: cenários de retransmissão



Cenário de ACK cumulativo

3 Geração de ACK [RFC 1122, RFC 2581]

Evento no receptor	Ação do receptor TCP
Segmento chega em ordem, não há lacunas, segmentos anteriores já aceitos	ACK retardado. Espera até 500 ms 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 detectado	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

3 Retransmissão rápida

- Com freqüência, o tempo de expiração é relativamente longo:
  - Longo atraso antes de reenviar um pacote perdido
- Detecta segmentos perdidos por meio de ACKs duplicados
  - Transmissor freqüentemente envia muitos segmentos *back-to-back*
  - Se o segmento é perdido, haverá muitos ACKs duplicados
- Se o transmissor recebe 3 ACKs para o mesmo dado, ele supõe que o segmento após o dado confirmado foi perdido:
  - Retransmissão rápida: reenvia o segmento antes de o temporizador expirar

3 Algoritmo de retransmissão rápida

```
event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase = y
                if (there are currently not-yet-acknowledged segments)
                    start timer
              }
            else {
                increment count of dup ACKs received for y
                if (count of dup ACKs received for y = 3) {
                    resend segment with sequence number y
                 }
```

ACK duplicado para um segmento já confirmado

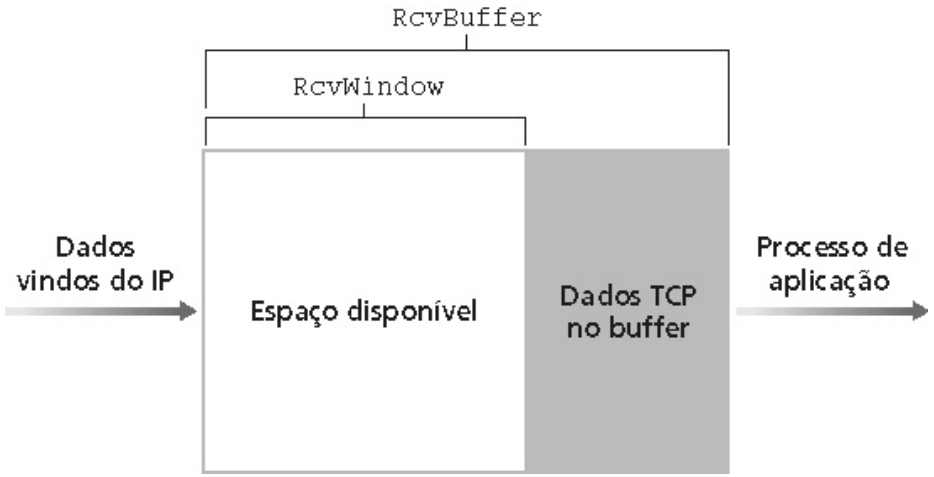
retransmissão rápida

3 Camada de transporte

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

3 TCP: controle de fluxo

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



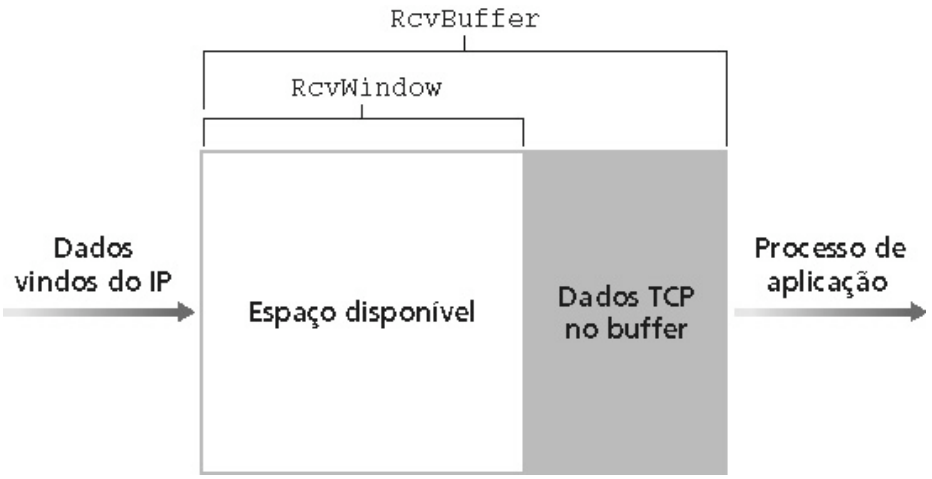
- Processos de aplicação podem ser lentos para ler o buffer

Controle de fluxo

Transmissor não deve esgotar os buffers de recepção enviando dados rápido demais

- Serviço de **speed-matching:** encontra a taxa de envio adequada à taxa de vazão da aplicação receptora

3 Controle de fluxo TCP: como funciona



- Receptor informa a área dispor incluindo valor **`RcvWindow`** nos segmentos
- Transmissor limita os dados não confinados ao **`RcvWindow`**
  - Garantia contra overflow no buffer do receptor

(suponha que o receptor TCP descarte segmentos fora de ordem)

- Espaço disponível no buffer

`= RcvWindow`

`= RcvBuffer-[LastByteRcvd - LastByteRead]`

3 Camada de transporte

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

3 Gerenciamento de conexão TCP

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

- Inicializar 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 SYNA
- Reconhece o SYN recebido
- Aloca buffers
- Especifica o número de seqüência inicial do servidor

Passo 3: sistema final cliente reconhece o SYNACK

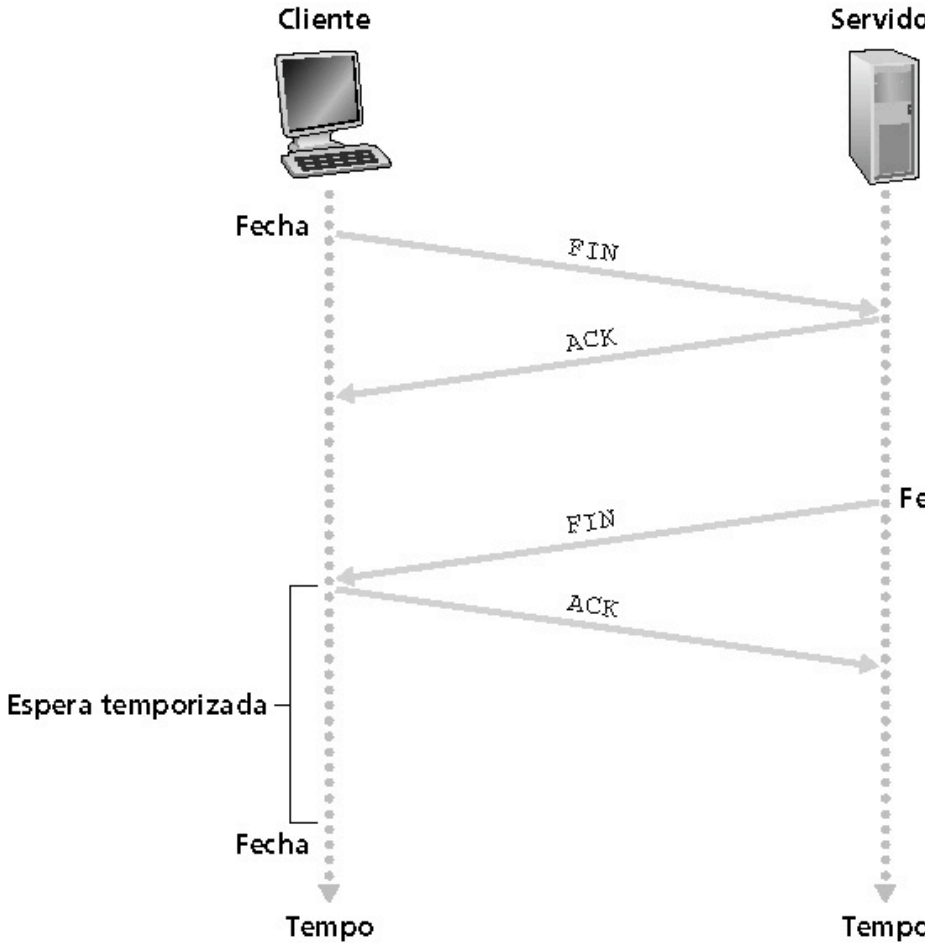
3 Gerenciamento de conexão TCP

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



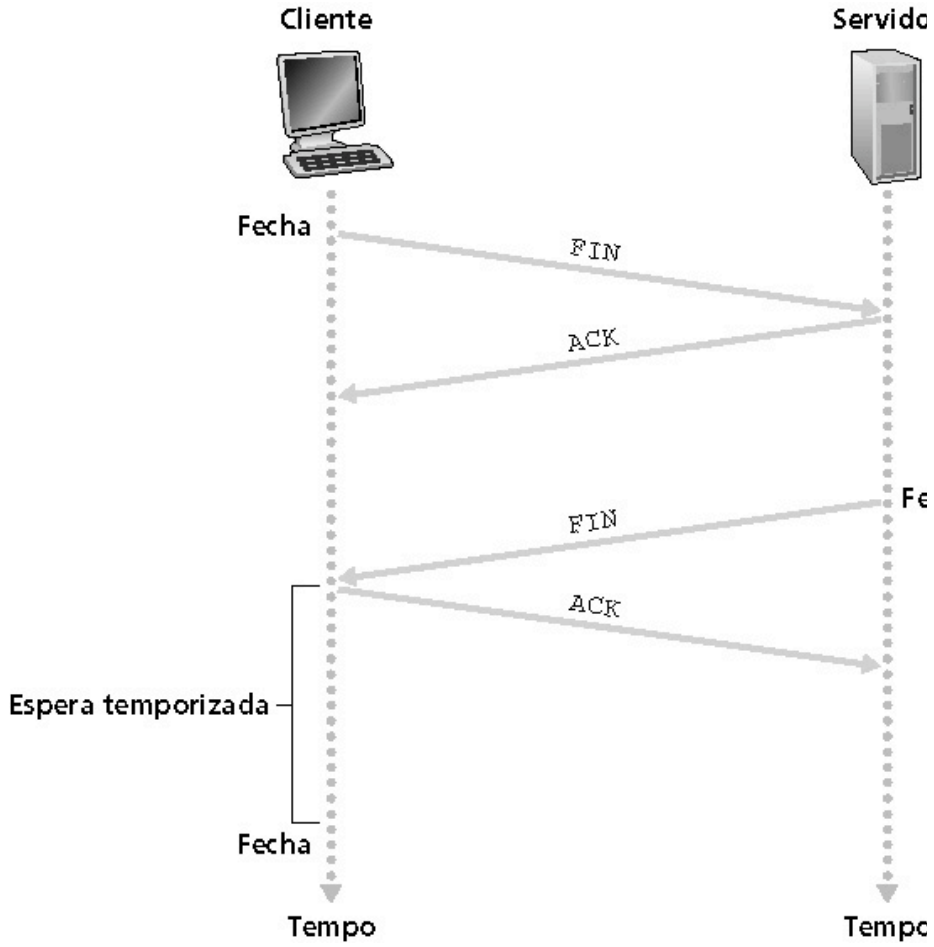
3 Gerenciamento de conexão TCP

Passo 3: cliente recebe FIN, responde com ACK

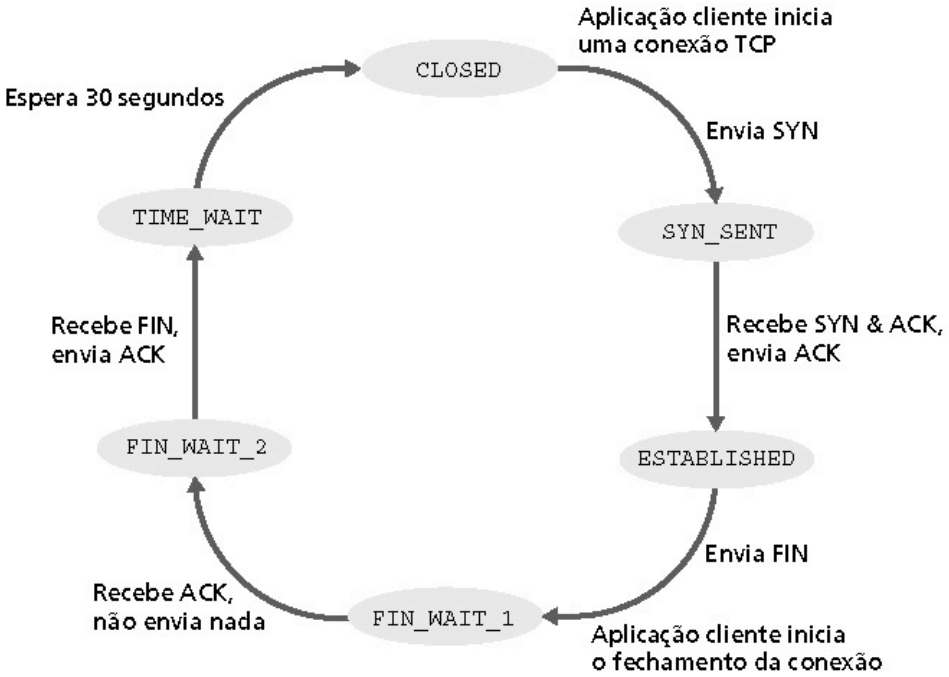
• Entra “espera temporizada” - vai responder com ACK a FINs recebidos

Passo 4: servidor, recebe ACK
Conexão fechada

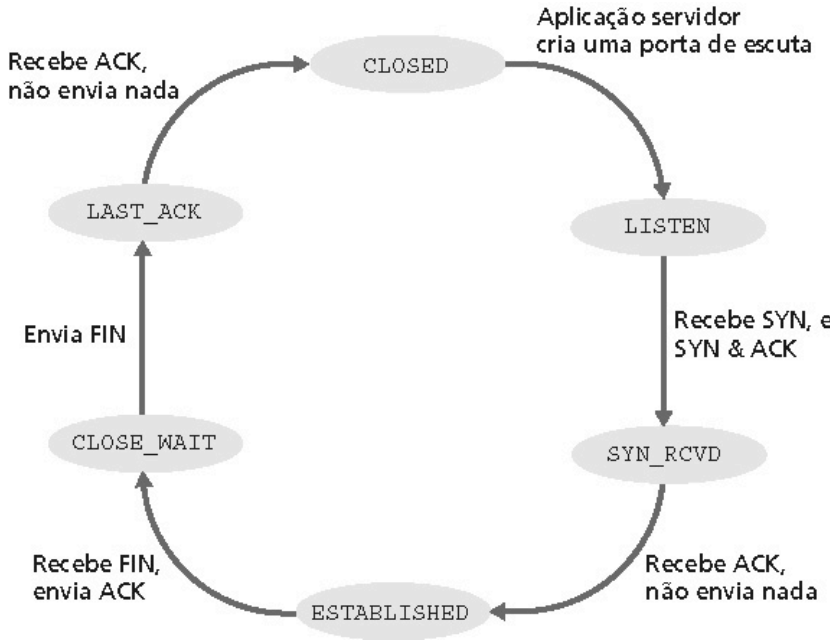
Nota: com uma pequena modificação, pode-se manipular FINs simultâneos



3 Gerenciamento de conexão TCP



Estados do cliente



Estados do servidor

3 Camada de transporte

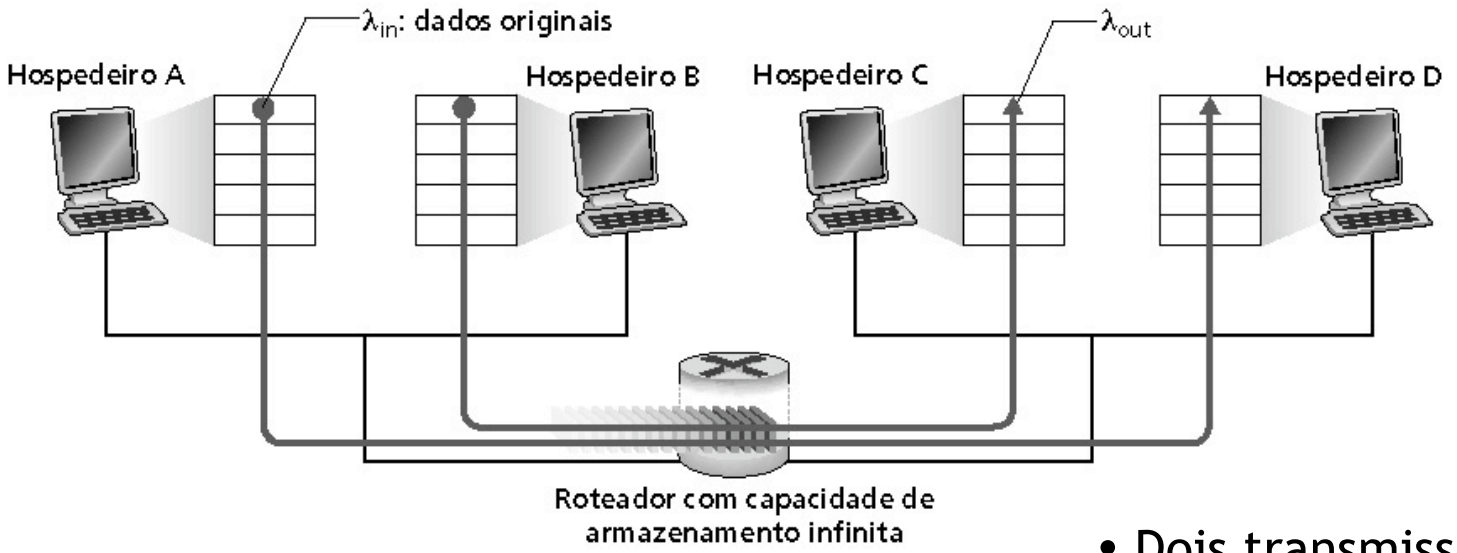
- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado à conexão: UDP
- 3.4 Princípios de transferência confiável de dados
- 3.5 Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo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

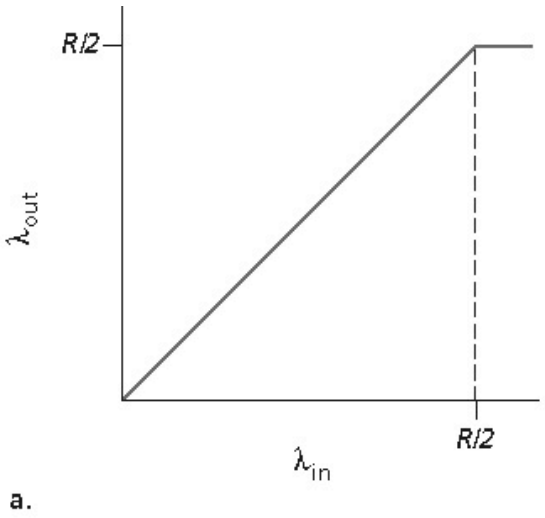
3 Princípios de controle de congestionamento

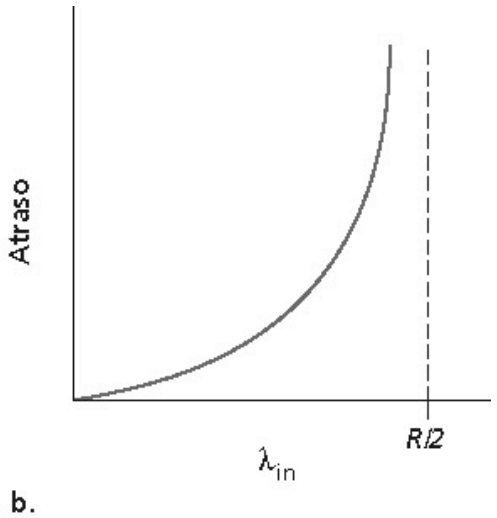
Congestionamento:

- Informalmente: “muitas fontes enviando dados acima da capacidade da rede de 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!

3 Causas/custos do congestionamento: cenário 1



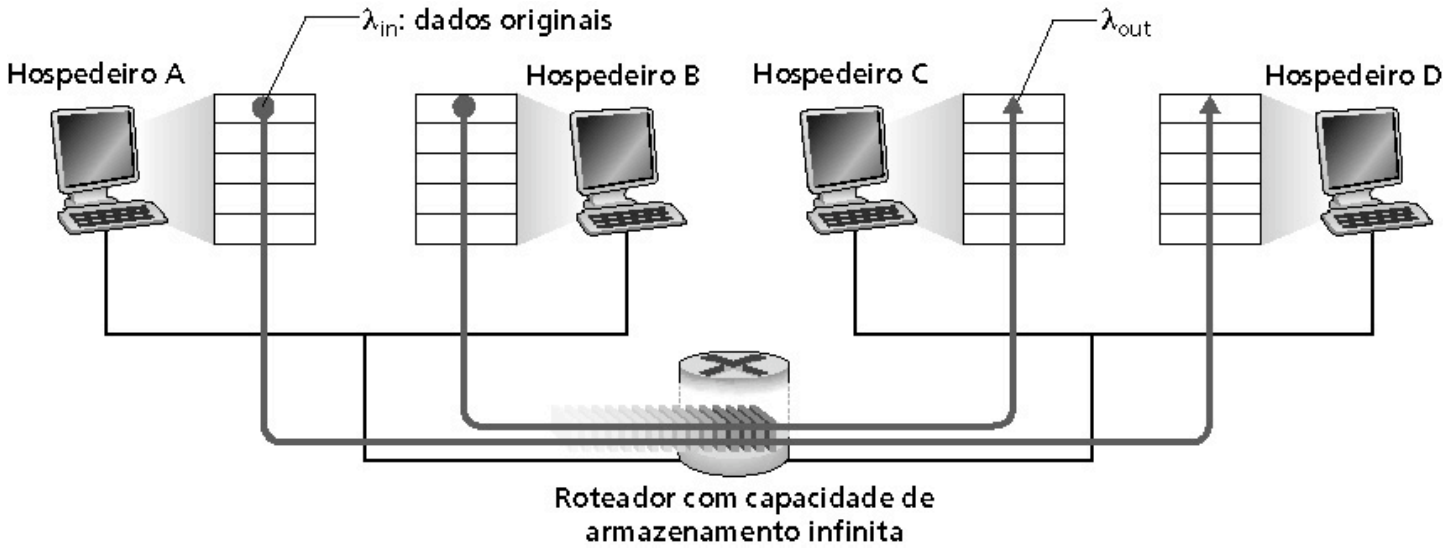




- Dois transmissores, do receptores
- Um roteador, buffers infinitos
- Não há retransmissão
- Grandes atrasos quand congestionado
- Máxima vazão alcançá

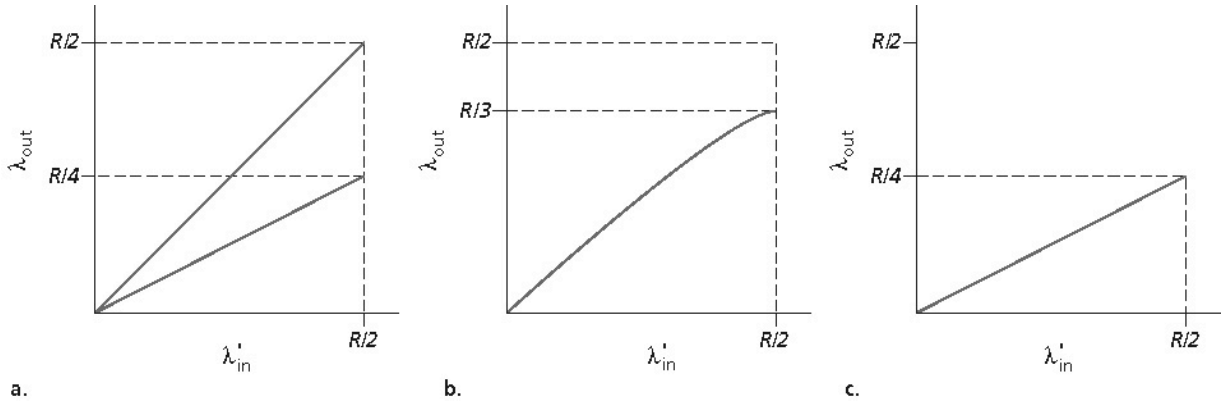
3 Causas/custos do congestionamento: cenário 2

- Um roteador, buffers **finitos**
- Transmissor reenvia pacotes perdidos



3 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 cas perfeito) para o mesmo λ_{out}



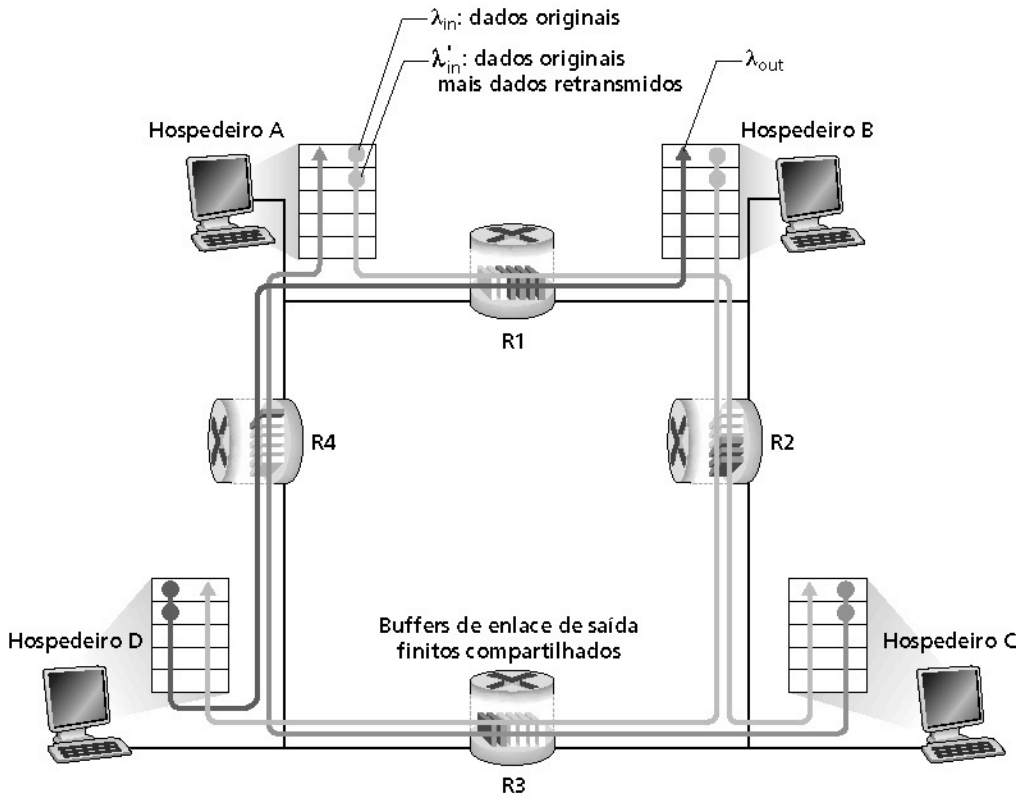
"custos" do congestionamento:

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

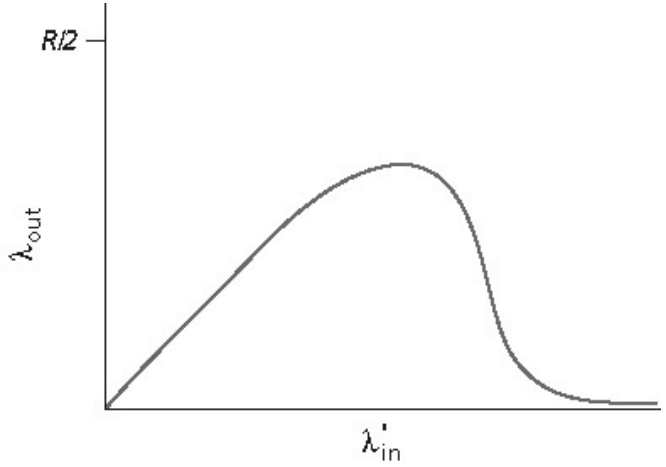
3 Causas/custos do congestionamento: cenário 3

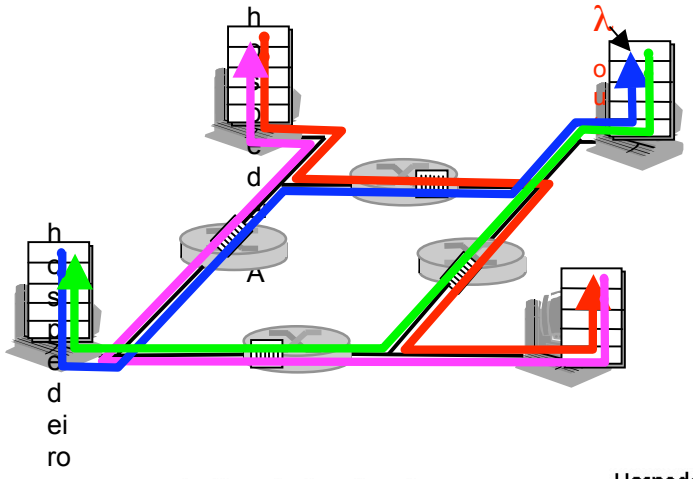
- Quatro transmissores
- Caminhos com múltiplos saltos
- Temporizações/retransmissões

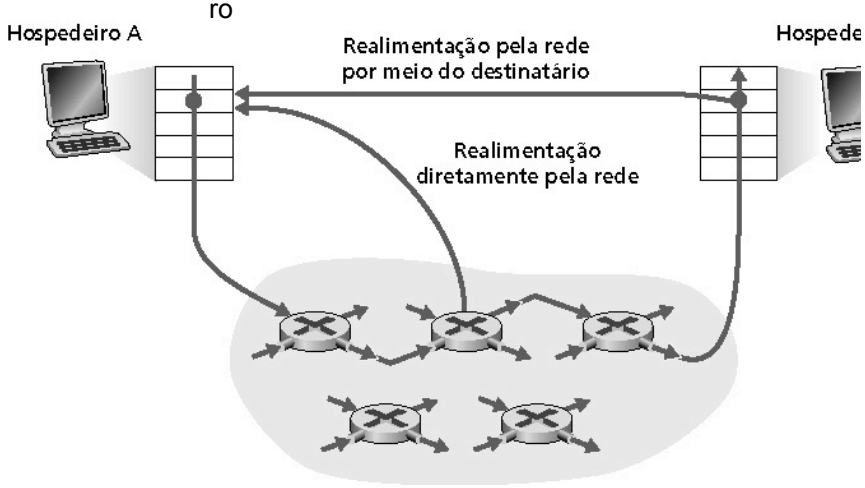
P.: O que acontece quando λ_{in} e λ aumentam?



3 Causas/custos do congestionamento: cenário 3







02-068
AW/Kurose and Ross
Computer Networking
KR 03.47 ar2
15p6 Wide x 11p Deep
2/c
05/14/02GM 6/03/02GM

Outro "custo" do congestionamento:

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

3 Abordagens do produto 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

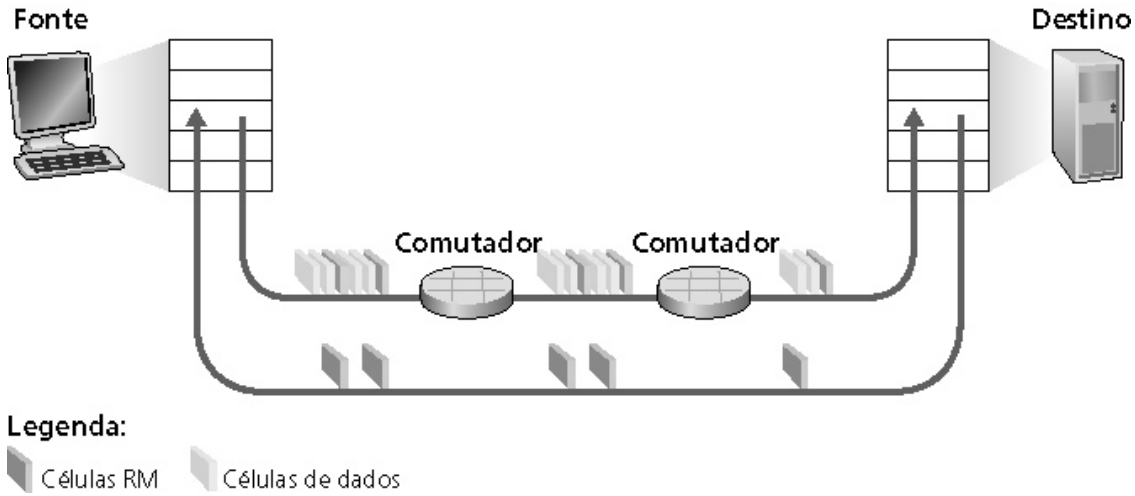
Controle de congestionamento assistido pela rede:
- Roteadores enviam informações para os sistemas finais
  - Bit único indicando o congestionamento (SNA, DECbit, TCP/IP ECN, ATM)
  - Taxa explícita do transmissor poderia ser enviada

3 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 aumenta a taxa (congestionamento leve)
  - CI bit: indicação de congestionamento
- As células RM são devolvidas ao transmissor pelo receptor, com os bits de indicação intactos

3 Estudo de caso: controle de congestionamento do servidor do serviço ATM ABR



- **Campo ER (explicit rate) de dois bytes nas células RM**
  - Switch congestionado pode reduzir o valor de ER nas células
  - O transmissor envia dados de acordo com essa vazão mínima suportada no caminho
- **Bit EFCI nas células de dados: marcado como 1 pelos switches congestionados**
  - Se a célula de dados que precede a célula RM tem o bit EFCI setado, o receptor marca o bit CI na célula RM devolvida

3 Camada de transporte

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

3 TCP: controle de congestionamento

- Controle fim-a-fim (sem assistência da rede)
- Transmissor limita a transmissão:
  `LastByteSent-LastByteAcked`
  `≤ CongWin`
- Aproximadamente,

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- **`CongWin`** é dinâmico, função de congestionamento das redes detectadas

Como o transmissor detecta o congestionamento?

- Evento de perda = tempo de confirmação • *ou* 3 ACKs duplicados
  Transmissor TCP reduz a taxa (**`CongWin`**) após o evento de perda

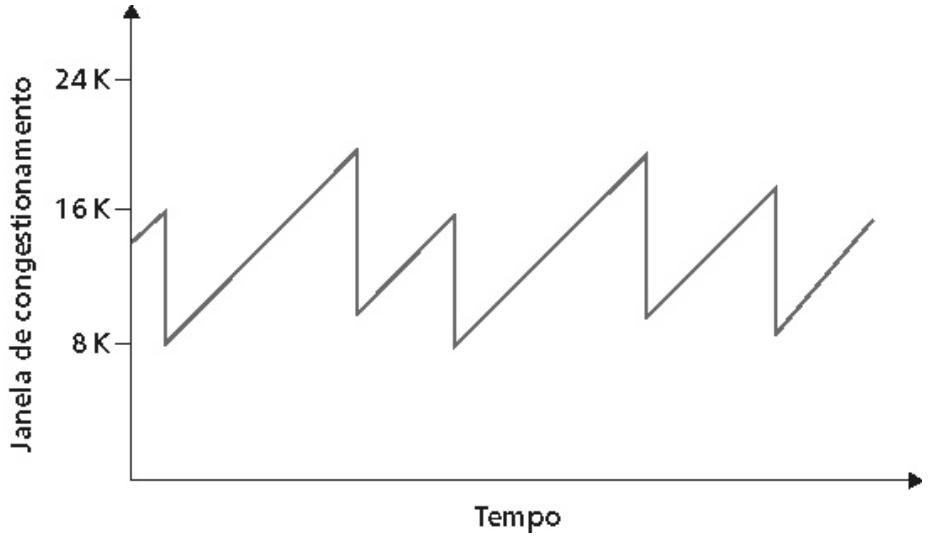
Três mecanismos:

- AIMD
- Partida lenta
- Reação a eventos de esgotamento de temporização

3 TCP AIMD

Redução multiplicativa: diminui o **`CongWin`** pela metade após o evento de perd

Aumento aditivo: aumenta o **`CongWin`** com 1 MSS a cada RTT na ausência de eventos de perda: **probing**



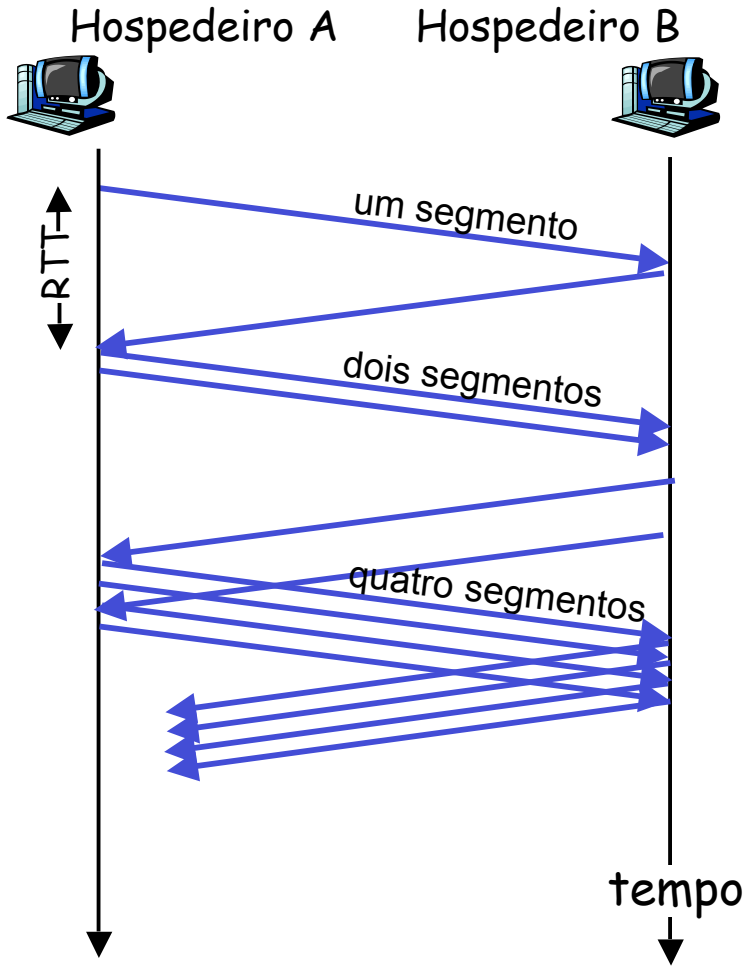
conexão TCP de longa-vida

3 TCP Partida lenta

- Quando a conexão começa, **CongWin** = 1 MSS
  - Exemplo: MSS = 500 bytes e RTT = 200 milissegundos
  - Taxa inicial = 20 kbps
- Largura de banda disponível pode ser >> MSS/RTT
  - Desejável aumentar rapidamente até a taxa respeitável
- Quando a conexão começa, a taxa aumenta rapidamente de modo exponencial até a ocorrência do primeiro evento de perda

3 TCP Partida lenta

- Quando a conexão começa, a taxa aumenta rapidamente de modo exponencial até a ocorrência do primeiro evento de perda :
  - Dobra o **`CongWin`** a cada RTT
  - Faz-se incrementando o **`CongWin`** para cada ACK recebido
- Sumário: taxa inicial é lenta mas aumenta de modo exponencialmente rápido



3 Refinamento

- Após 3 ACKs duplicados:
  - **`CongWin`** é cortado pela metade
  - Janela então cresce linearmente
- Mas após evento de tempo de confirmação:
  - **`CongWin`** é ajustado para 1 MSS;
  - A janela então cresce exponencialmente até um limite, então cresce linearmente

Filosofia

- 3 ACKs indica que a rede é capaz de entregar alguns segmentos
- Tempo de confirmação antes dos 3 ACKs duplicados é “mais alarmante”

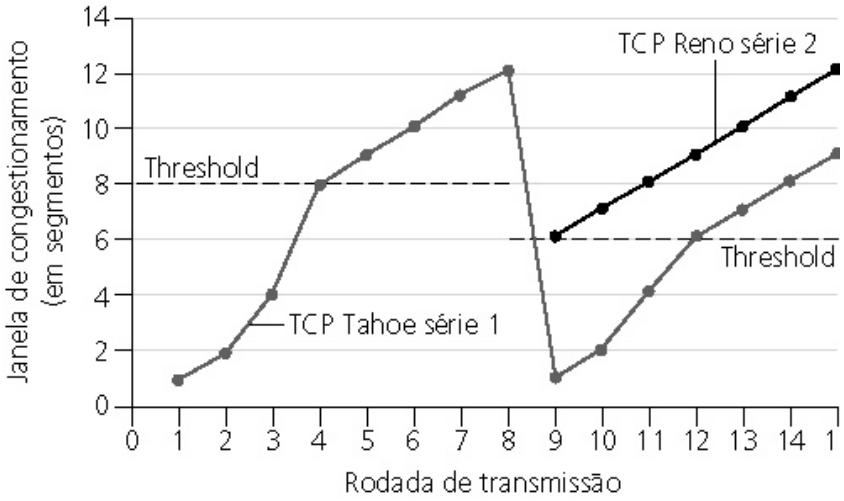
3 Refinamento

P.: Quando o aumento exponencial deve tornar-se linear?

R.: Quando **`CongWin`** obtiver 1/2 do seu valor antes do tempo de confirmação.

Implementação:

- Limite variável
- No evento de perda, o limiar é ajustado para 1/2 do CongWin logo antes do evento de perda



3 Resumo: controle de congestionamento TCP

- Quando **CongWin** está abaixo do limite (**Threshold)**, o transmissor em fase de slow-start, a janela cresce exponencialmente.
- Quando **CongWin** está acima do limite (**Threshold**, o transmissor em fase de congestion-avoidance, a janela cresce linearmente.
- Quando ocorrem três ACK duplicados, o limiar (**Threshold**) é ajustado em **CongWin/2** e **CongWin** é ajustado para **Threshold**.
- Quando ocorre tempo de confirmação, o **Threshold** é ajustado para **CongWin/2** e o **CongWin** é ajustado para 1 MSS.

3 TCP sender congestion control

Evento	Estado	Ação do transmissor TCP	Comentário
ACK recebido para dado previamente não confirmado	partida lenta (SS)	CongWin = CongWin + MSS, If (CongWin > Threshold) ajusta estado para "prevenção de congestionamento"	Resulta em dobrar o CongWin a cada RTT
ACK recebido para dado previamente não confirmado	prevenção de congestiona-mento (CA)	CongWin = CongWin + MSS * (MSS/CongWin)	Aumento aditivo, resulta no aumento do CongWin em 1 MSS a cada RTT
Evento de perda detectado por três ACKs duplicados	SS or CA	Threshold = CongWin/2, CongWin = Threshold, ajusta estado para "prevenção de congestionamento"	Recuperacao rápida, implementando redução multiplicativa o CongWin não cairá abaixo de 1 MSS.
Tempo de confirmação	SS or CA	Threshold = CongWin/2, CongWin = 1 MSS, ajusta estado para "partida lenta"	Entra em partida lenta
ACK duplicado	SS or CA	Incrementa o contador de ACK duplicado para o segmento que está sendo confirmado	CongWin e Threshold nã mudam

3 TCP **throughput**

- O que é **throughout** médio do TCP como uma função do tamanho da janela e do RTT?
  Ignore a partida lenta
- Deixe W ser o tamanho da janela quando ocorre perda
- Quando a janela é **W**, o **throughput** é W/RTT
- Logo após a perda, a janela cai para W/2, e o **throughput** para W/2RTT
- **Throughout** médio: 0,75 W/RTT

3 Futuro do TCP

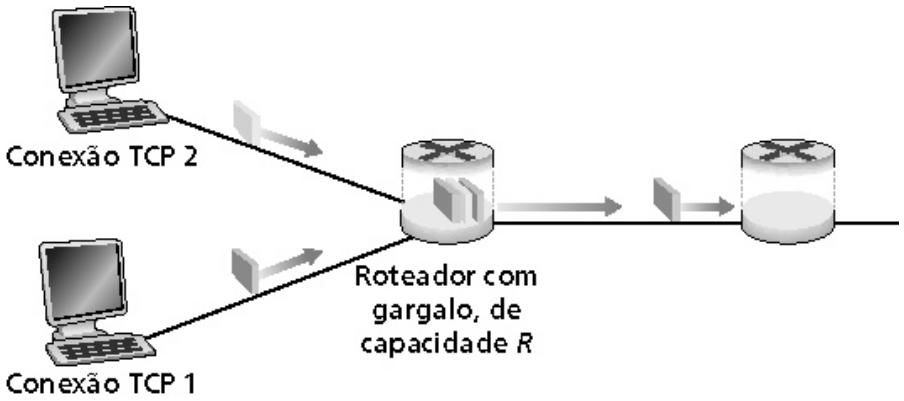
- Exemplo: segmento de 1500 bytes, RTT de 100 ms, deseja 10 Gbps de *throughput*
- Requer tamanho de janela W = 83,333 para os segmentos em trânsito
- Throughput em termos da taxa de perda:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ $L = 2 \cdot 10^{-10}$ Uau!
- São necessárias novas versões de TCP para alta velocidade!

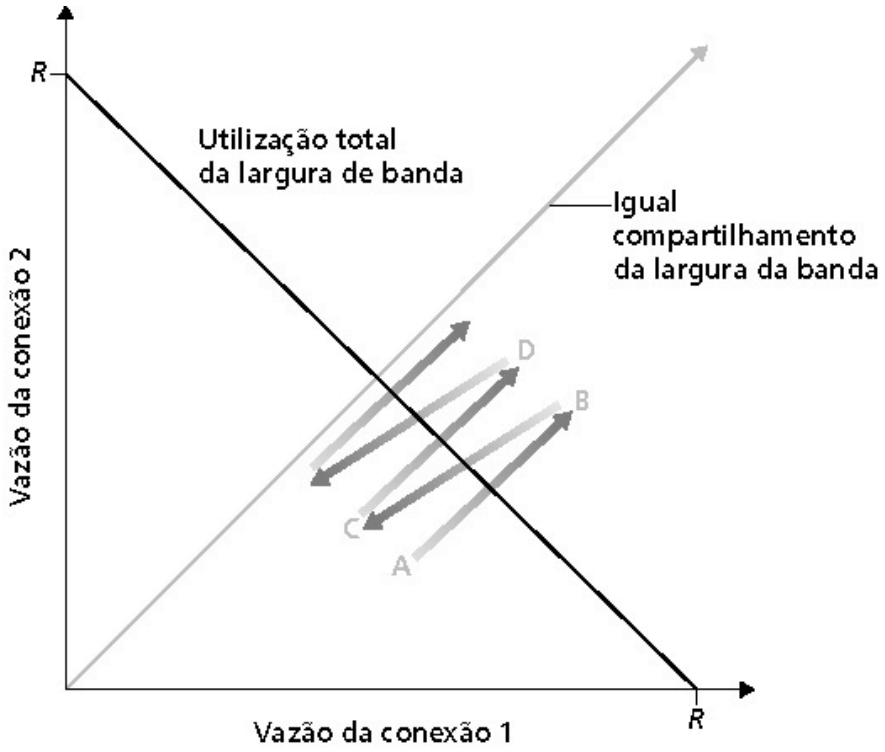
3 Eqüidade do TCP

Objetivo de eqüidade: se K sessões TCP compartilham o mesmo enlace do gargalo com largura de banda R, cada uma deve ter taxa média de R/K



3 Por que 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



perda: reduz janela por um fator de

prevenção de congestionamento: aumento aditivo

perda: reduz janela por um fator de

prevenção de congestionamento: aumento aditivo

3 Eqüidade

Eqüidade e UDP

- Aplicações multimídia normalmente não usam TCP
  - Não querem a taxa estrangulada pelo controle de congestionamento
- Em vez disso, usam UDP:
  - Trafega áudio/vídeo a taxas constantes, toleram perda de pacotes

  Área de pesquisa: TCP amigável

Eqüidade e conexões TCP paralelas

- Nada previne as aplicações de abrirem conexões paralelas entre 2 hospedeirc
- Web browsers fazem isso
- Exemplo: enlace de taxa R suportando 9 conexões;
  - Novas aplicações pedem 1 TCP, obtém taxa de R/10
  - Novas aplicações pedem 11 TCPs, obtém R/2!

3 TCP: modelagem de latência

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

Ignorando o congestionamento, o atraso é influenciado por:

- Estabelecimento de conexão TCP
- Atraso de transferência de dados
- Partida lenta

Notação, hipóteses:

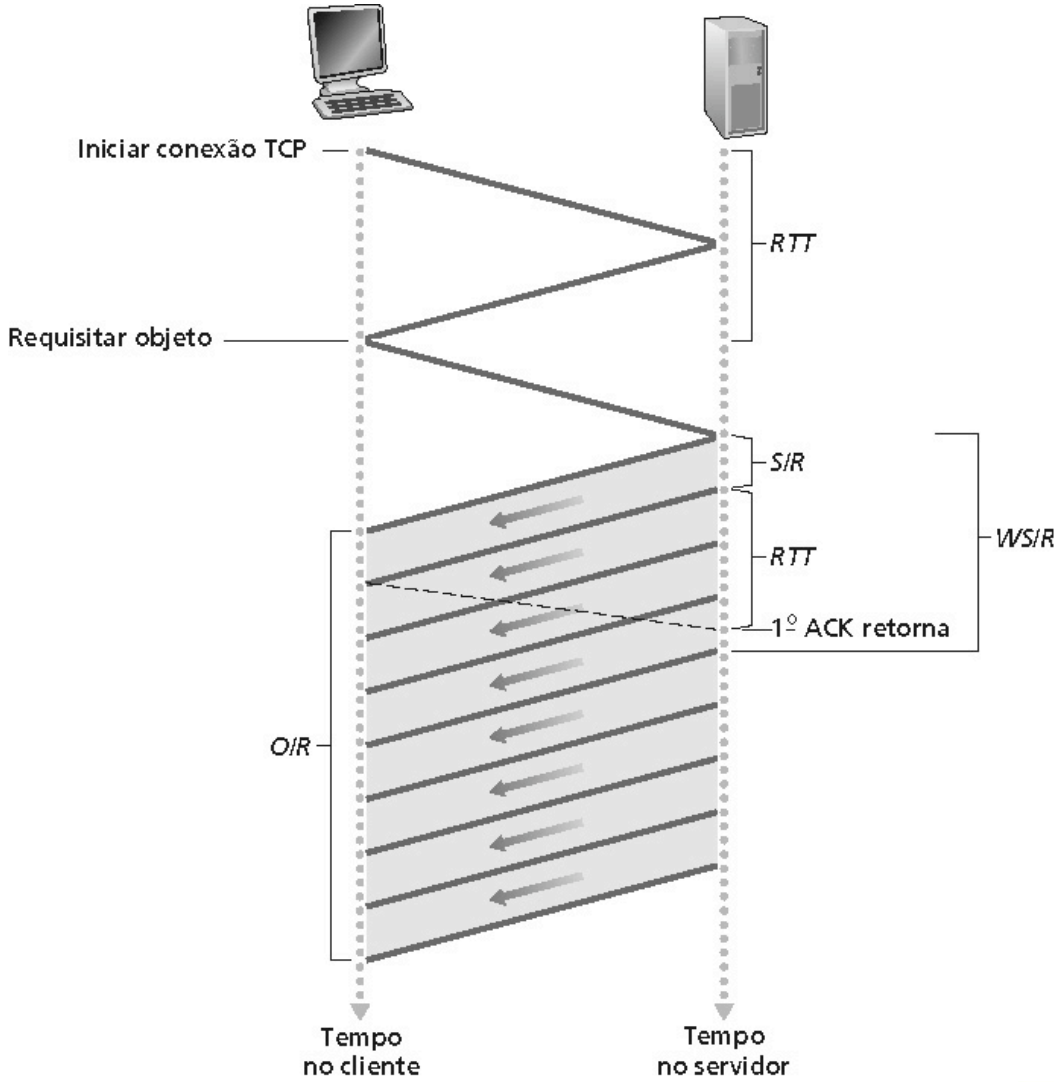
- Suponha um enlace entre o cliente e o servidor com taxa de dados R
- S: MSS (bits)
- O: tamanho do objeto (bits)
- Não há retransmissões (sem perdas e corrupção de dados)

Tamanho da janela:

- Primeiro suponha: janela de congestionamento fixa, W segmentos
- Então janela dinâmica, modelagem *partida lenta*

3 Janela de congestionamento fixa (1)

Primeiro caso:
WS/R > RTT + S/R: o ACK para o primeiro segmento na janela retorna antes do valor de janela dos dados enviados

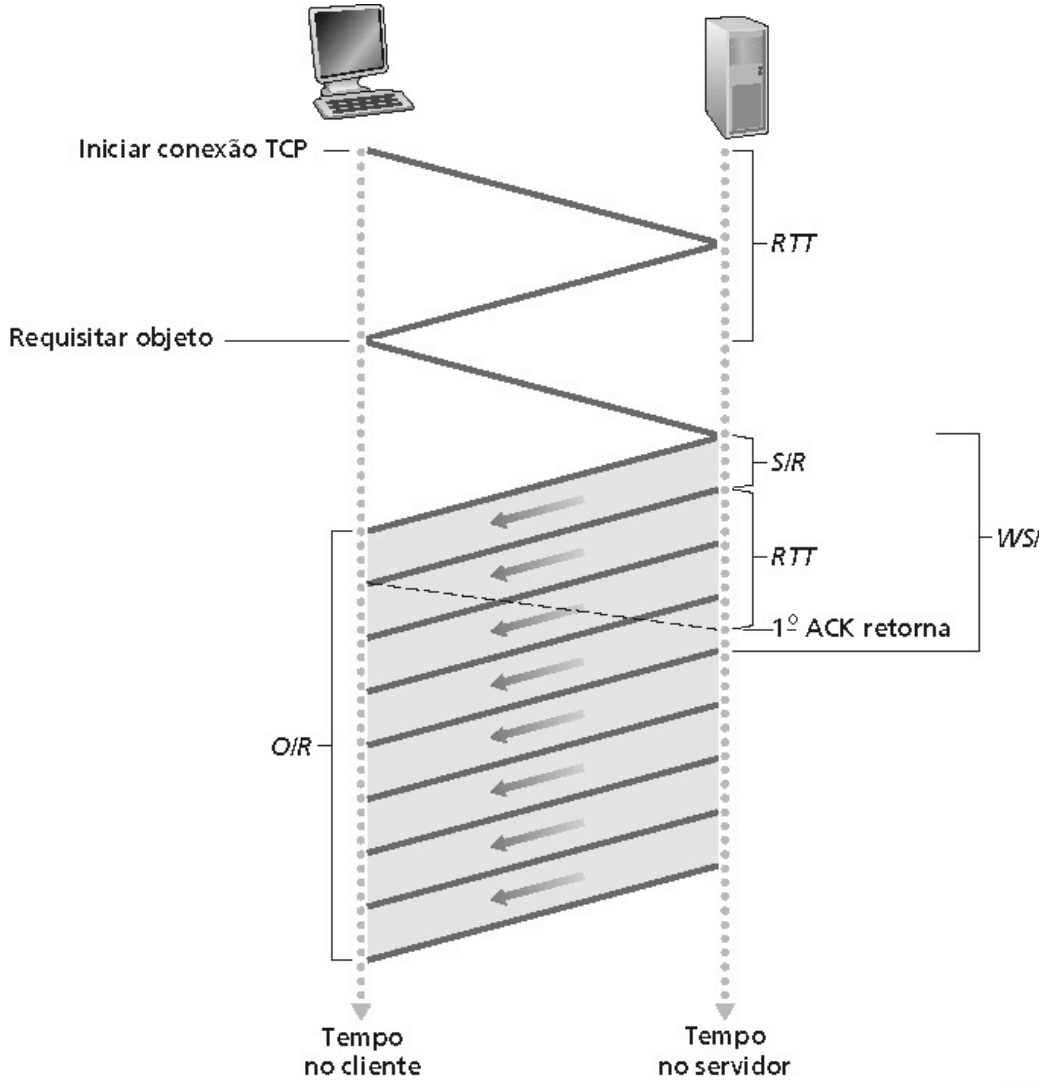


atraso = 2RTT + O/R

3 Janela de congestionamento fixa (2)

Segundo caso:

- WS/R < RTT + S/R: espera pelo ACK após enviar o valor da janela de dados



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

3 TCP Modelagem de latência: partida lenta (1)

- Agora suponha que a janela cresca de acordo com os procedimentos da fase partida lenta
- Vamos mostrar que a latência de um objeto de tamanho O é:

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

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

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

- Em que 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

3 TCP modelagem de latência: partida lenta (2)

Componentes do atraso:

- 2 RTT para estabelecimento de conexão e requisição
- O/R para transmitir um objeto
- Servidor com períodos inativos devido à **partida lenta**

Servidor inativo:

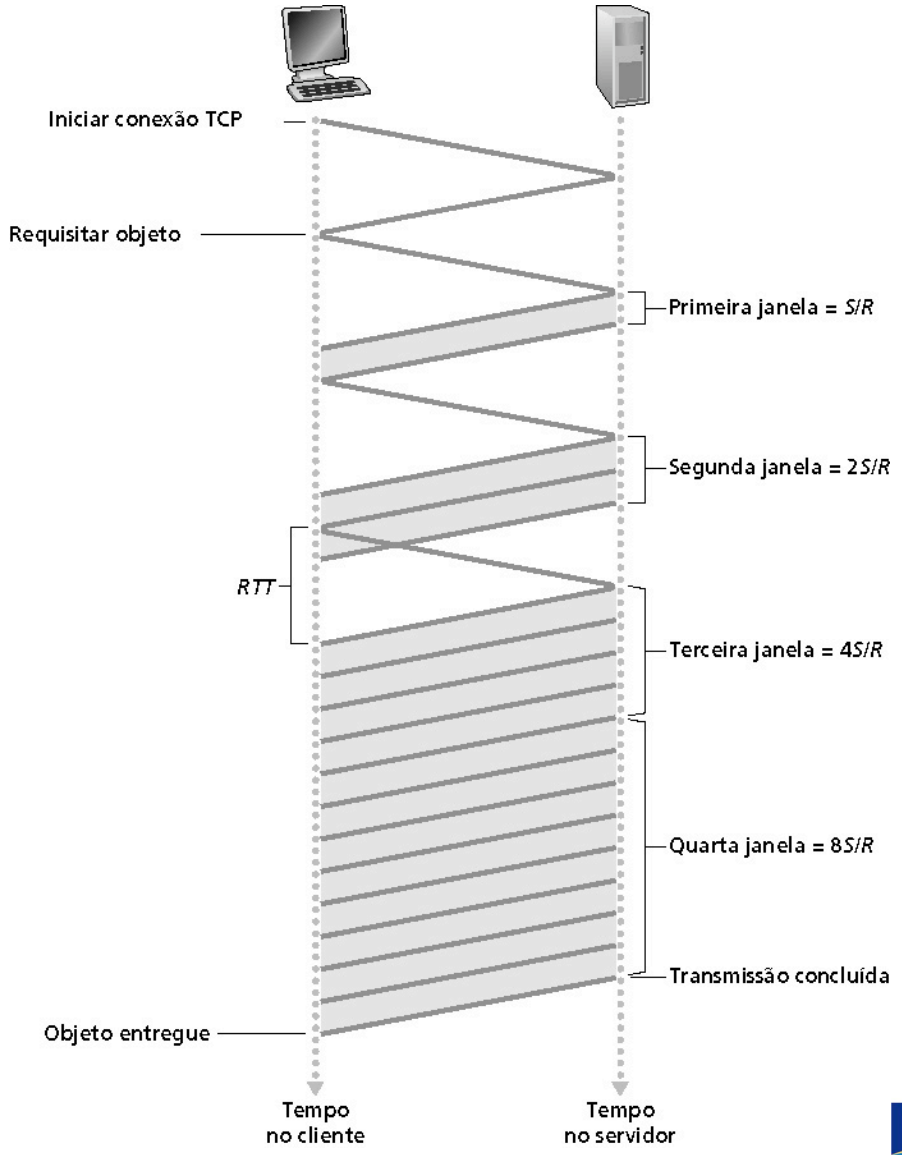
P = min{K-1,Q} vezes

Exemplo:

- O/S = 15 segmentos
- K = 4 janelas
- Q = 2

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

Servidor inativo P = 2 tempos



3 TCP modelagem de latência: partida lenta (3)

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

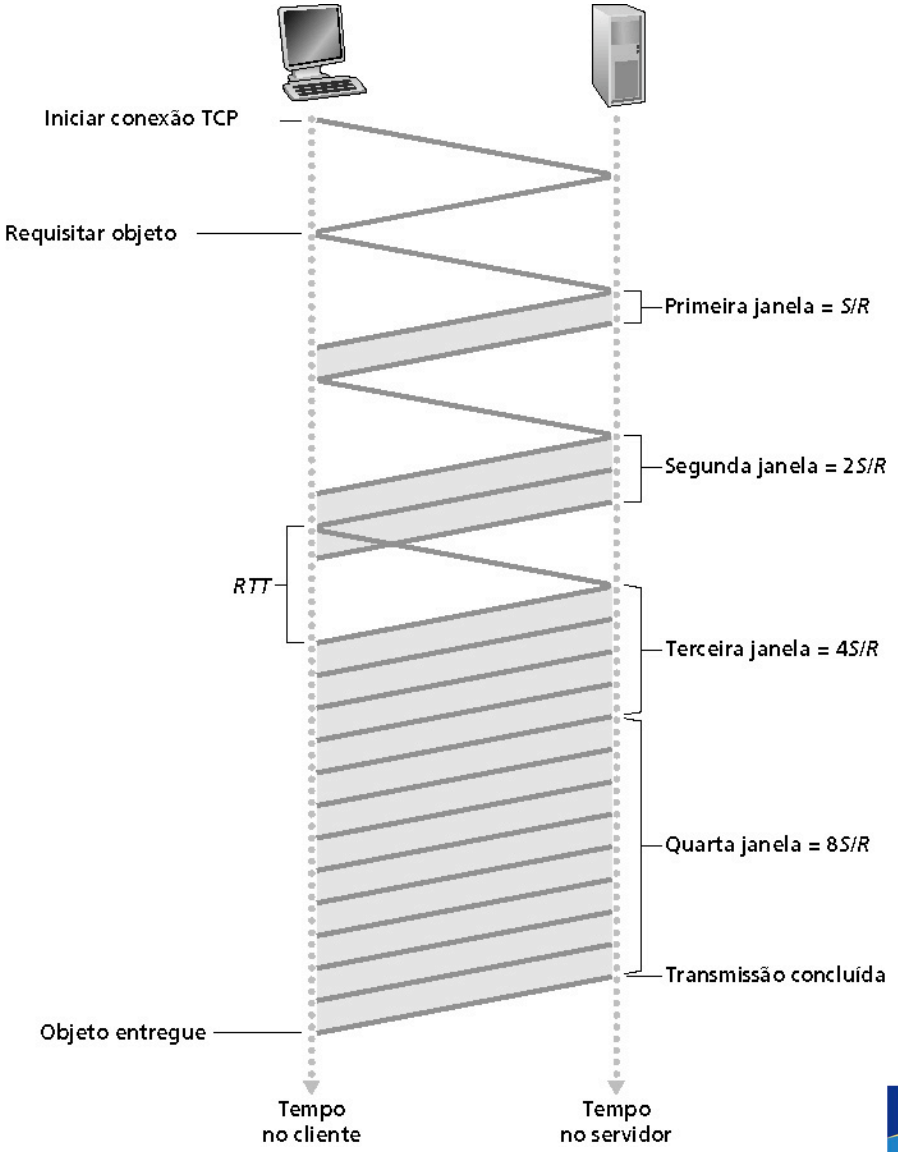
$2^{k-1}\frac{S}{R} =$ tempo para enviar a k-ésima janela

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

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

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

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



3 TCP modelagem de latência: partida lenta (4)

Lembre que K = número de janelas que cobrem um objeto.
Como calculamos o valor de K?

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

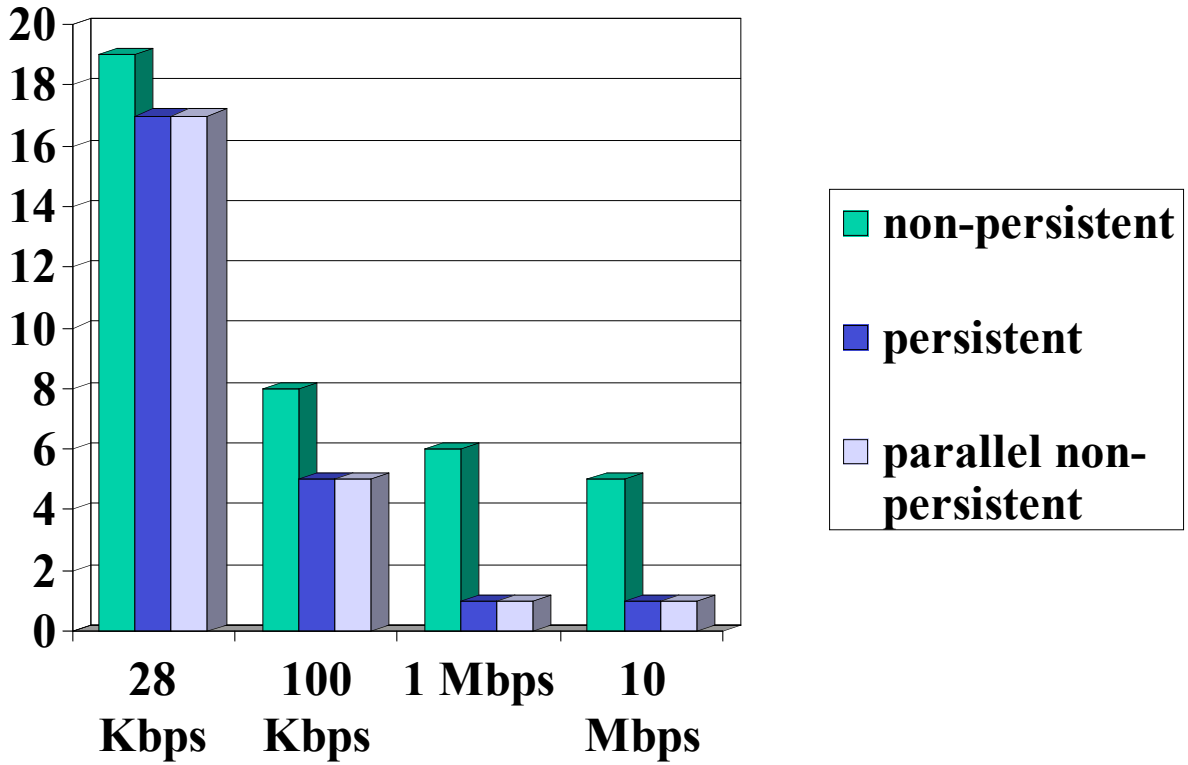
O cálculo do número Q, de inatividade por objeto de tamanho infinito, é similar (veja HW).

3 Modelagem HTTP

- Presuma que uma página Web consista em:
  - *1* página HTML de base (de tamanho *O* bit)
  - *M* imagens (cada uma de tamanho *O* bit)
- HTTP não persistente:
  - **M + 1** conexões TCP nos servidores
  - **Tempo de resposta = (M + 1)O/R + (M + 1)2RTT + soma dos períodos de inatividade**
- HTTP persistente:
  - **2 RTT** para requisitar e receber o arquivo HTML de base
  - **1 RTT** para requisitar e receber M imagens
  - **Tempo de resposta = (M + 1)O/R + 3RTT + soma dos períodos de inatividade**
- HTTP não persistente com X conexões paralelas
  - Suponha o inteiro M/X
  - 1 conexão TCP para o arquivo de base
  - M/X ajusta as conexão paralelas para imagens
  - **Tempo de resposta** = **(M + 1)O/R + (M/X + 1)2RTT + soma dos períodos de inatividade**

3 Tempo de resposta HTTP (em segundos)

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

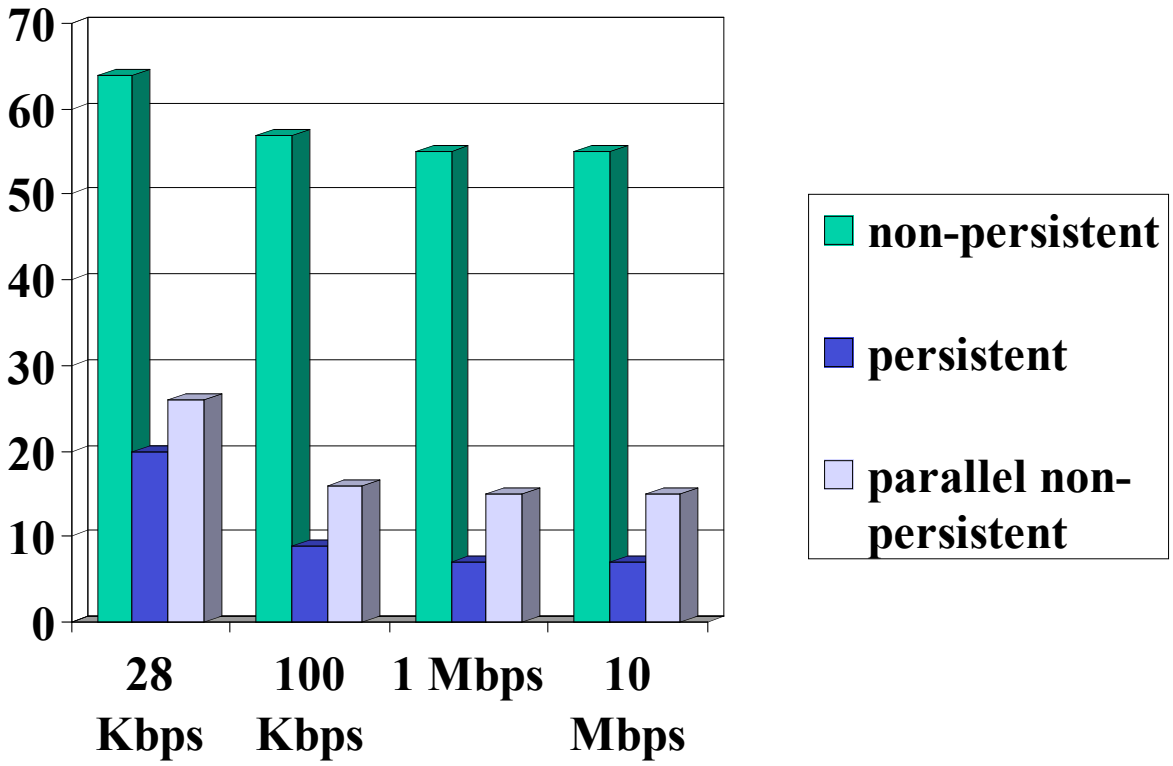


Para pouca largura de banda, tempo de conexão e resposta dominados pelo tempo de transmissão

Conexões persistentes oferecem pequena vantagem sobre as conexões paralelas

3 Tempo de resposta HTTP (em segundos)

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



Para longos RTT, o tempo de resposta é dominado por estabelecimento TCP e atrasos **partida lenta**. Conexões persistentes agora oferecem uma melhora. Importante: particularmente em redes com produto banda e atraso grande.

3 Resumo

- Princípios por trás dos serviços da camada de transporte:
  - Multiplexação/demultiplexação
  - Transferência de dados confiável
  - Controle de fluxo
  - Controle de congestionamento
- Instanciação e implementação na Internet
  - UDP
  - TCP

A seguir:

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