SSC0641
Redes de Computadores

Capítulo 3 - Camada de Transporte

Prof. Jó Ueyama

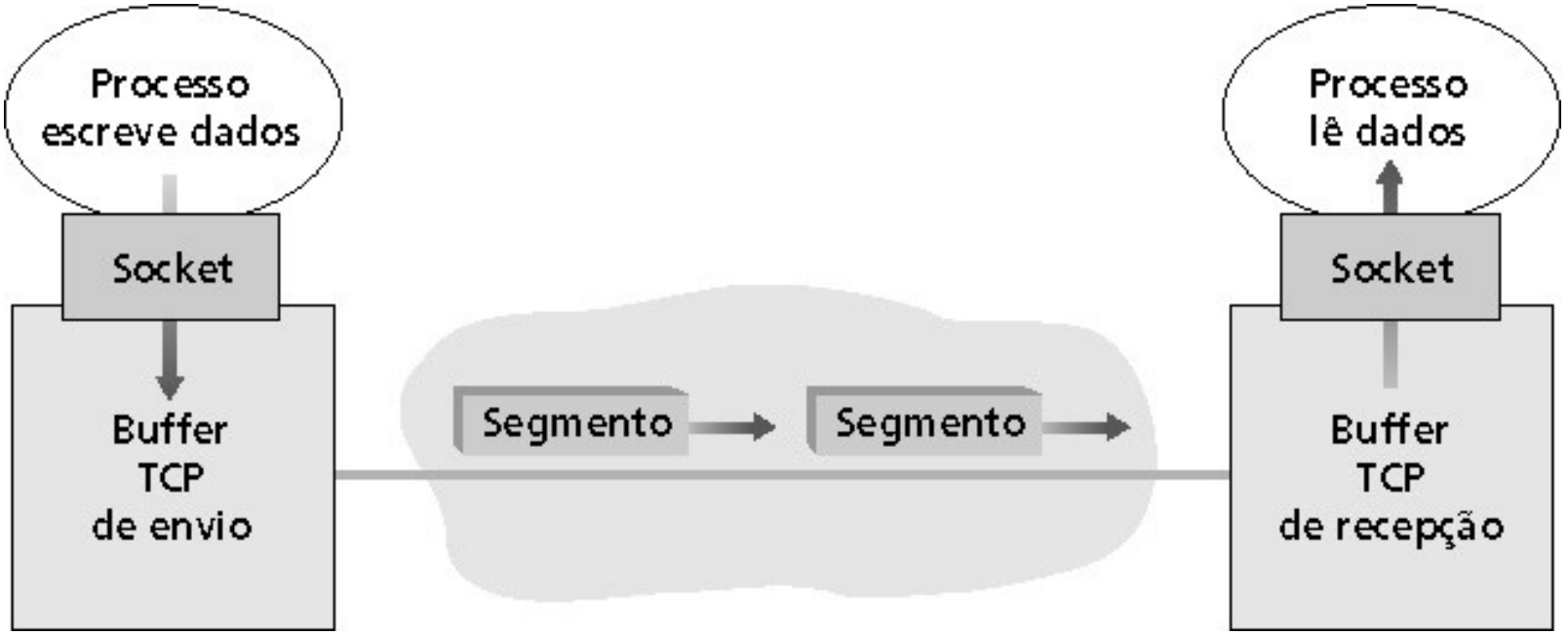
Março/2011

Transferência de Confiável de Dados

- Mecanismos:
  - Soma de verificação
  - Temporizador
  - Número de Seqüência
  - Mensagem de Reconhecimento
  - Mensagem de Reconhecimento Negativo
  - Janela

TCP

- Transmission Control Protocol.
- RFCs: 793, 1122, 1323, 2018, 2581.



TCP: Visão Geral

- Fim a fim:
  - Estados de conexão inteiramente nos dois sistemas finais e não nos roteadores
- Orientado à conexão:
  - apresentação: troca de mensagens de controle;
  - inicia o estado do transmissor e do receptor antes da troca de dados.
- Confiável;
- Full-duplex:
  - Transmissão bidirecional na mesma conexão.

TCP: Visão Geral

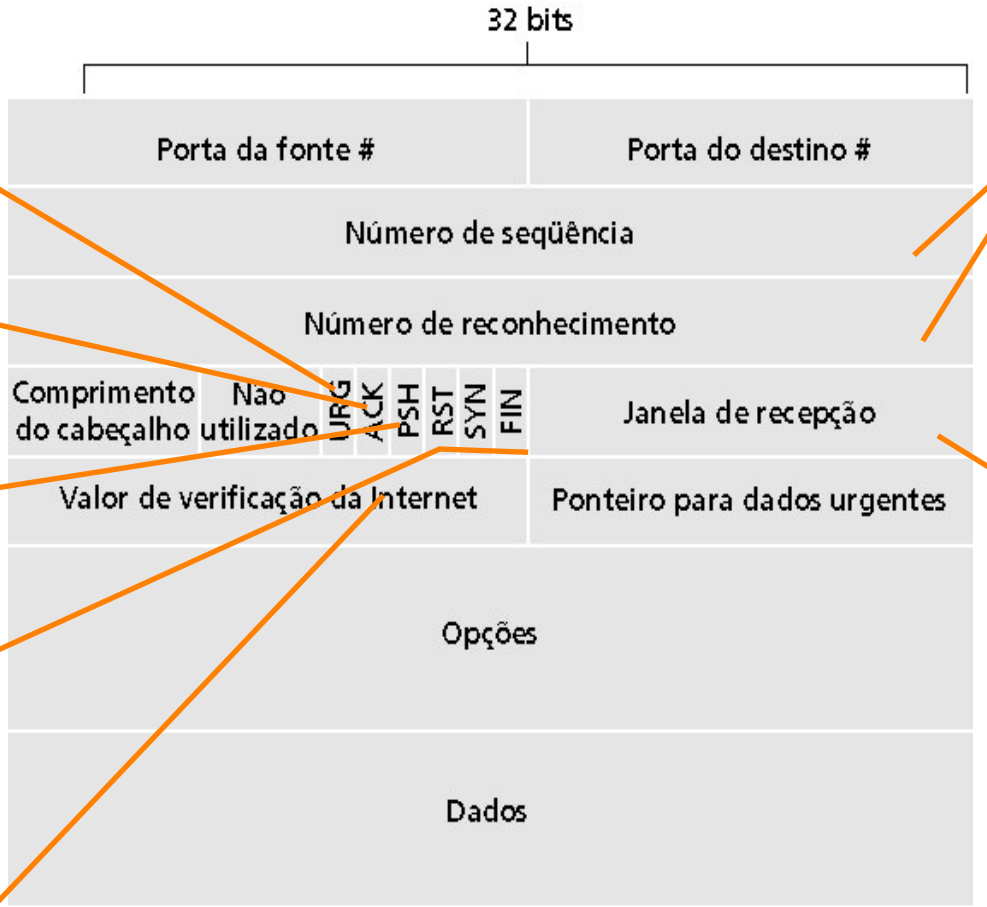
- Buffers de transmissão e de recepção:
  - Stream seqüencial de bytes;
  - MSS (maximum segment size): quantidade máxima de dados da camada de aplicação.
  - MTU (maximum transmission unit): tamanho máximo do quadro (camada de enlace) que remetente pode enviar.
    - Exemplos: 1460 bytes, 536 bytes.
  - Controle de fluxo:
    - Transmissor não esgota a capacidade do receptor.

TCP: Visão Geral

- Suporte Paralelismo:
  - Múltiplos segmentos sem confirmação;
  - Controle de congestionamento e de fluxo definem tamanho da janela.
- Applet ilustrando controle de fluxo:

http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/flow/flowcontrol.html

Estrutura do Segmento TCP



URG: dados urgentes (pouco usados)

ACK: campo de ACK é válido

PSH: envio de dados urgente p/ aplicação

RST, SYN, FIN: estabelecer conexão (comandos de criação e término)

Internet checksum (como no UDP)

contagem por bytes de dados (não segmentos!)

número de bytes receptor está pronto para aceitar

Conexão TCP

- Transmissor estabelece conexão com o receptor antes de trocar segmentos de dados .
- Inicializar variáveis:
  - Números de sequência
  - 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();
```

TCP: Estabelecimento de Conexão

Three way handshake:

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

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.

3: sistema final cliente reconhece o ACK.

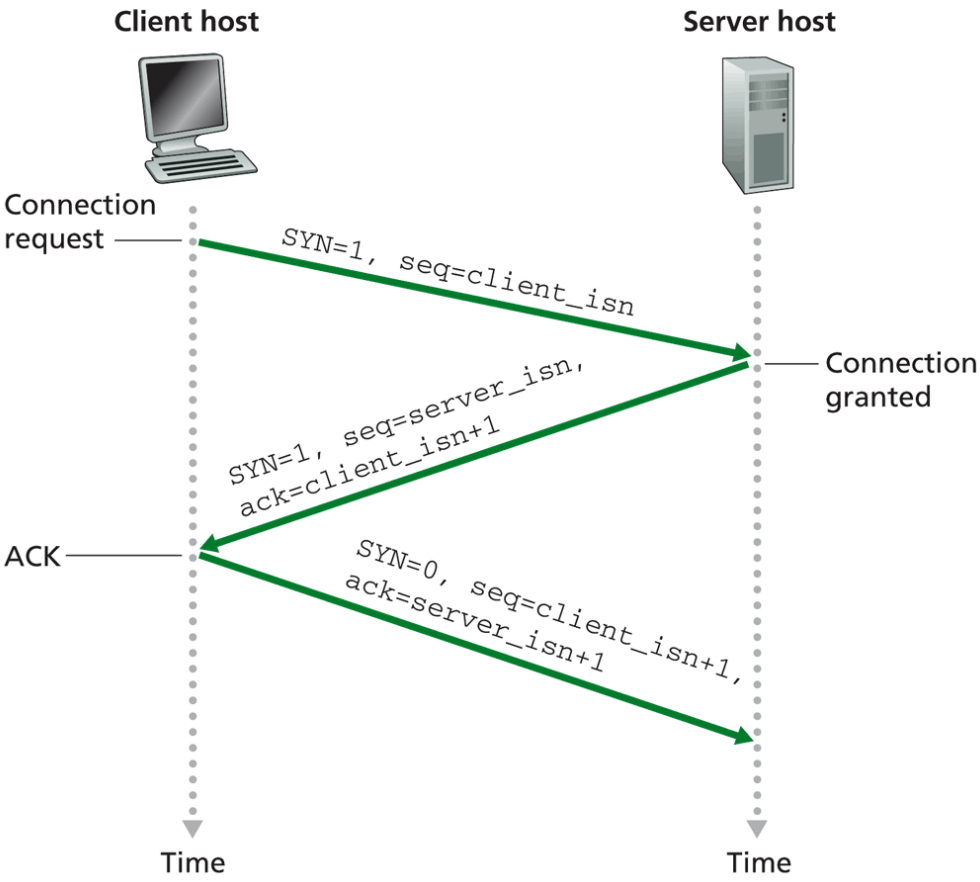


Figure 3.38 ♦ TCP three-way handshake: segment exchange

TCP: Fechando a Conexão

cliente fecha o socket:_
clientSocket.close();

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

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

3: cliente recebe FIN, responde com ACK.

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

4: servidor, recebe ACK Conexão fechada.

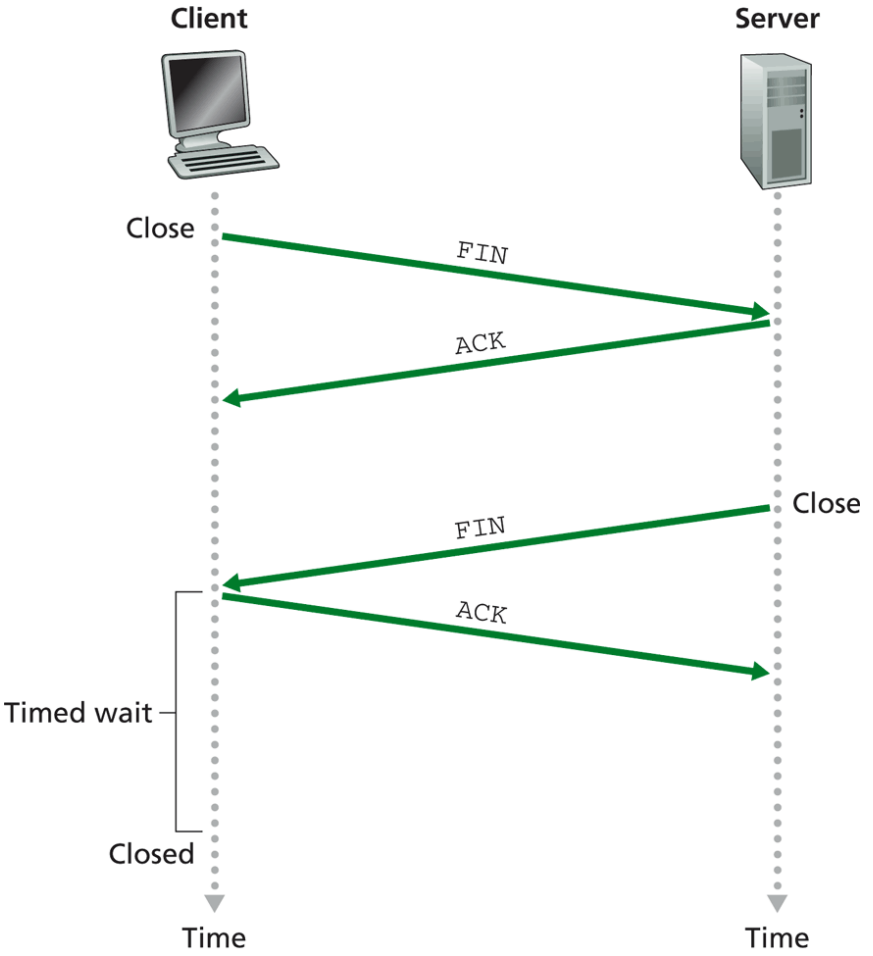
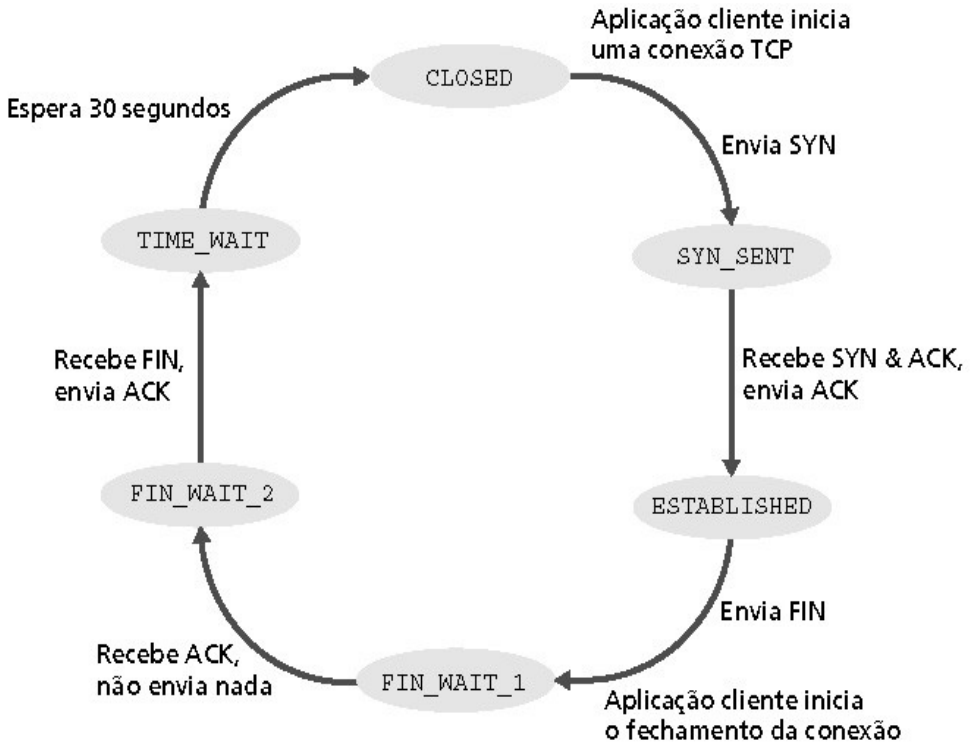
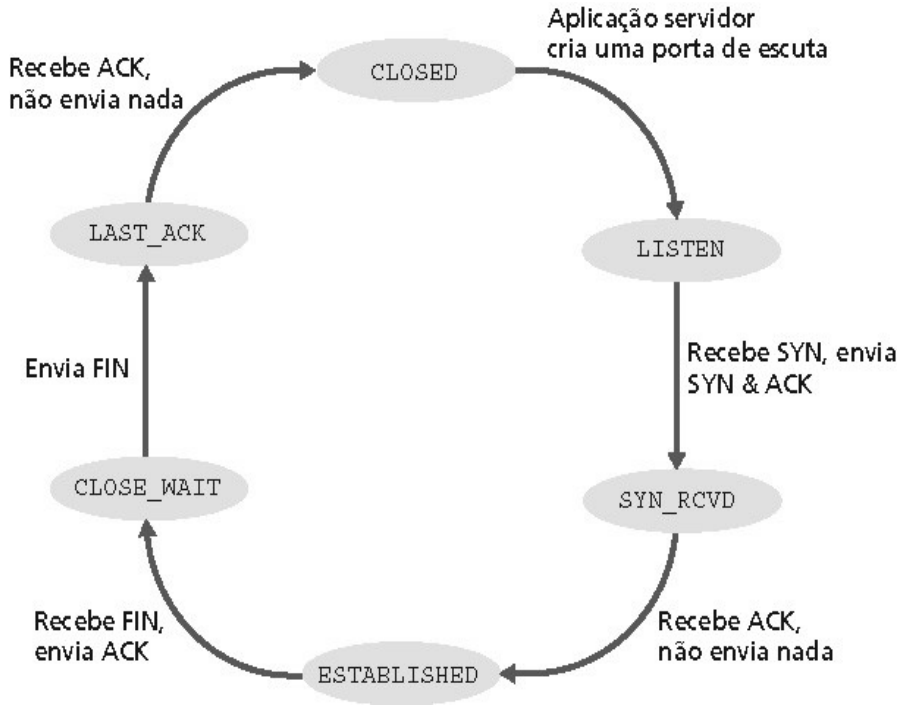


Figure 3.39 ♦ Closing a TCP connection

TCP: Máquina de Estados



Estados do cliente

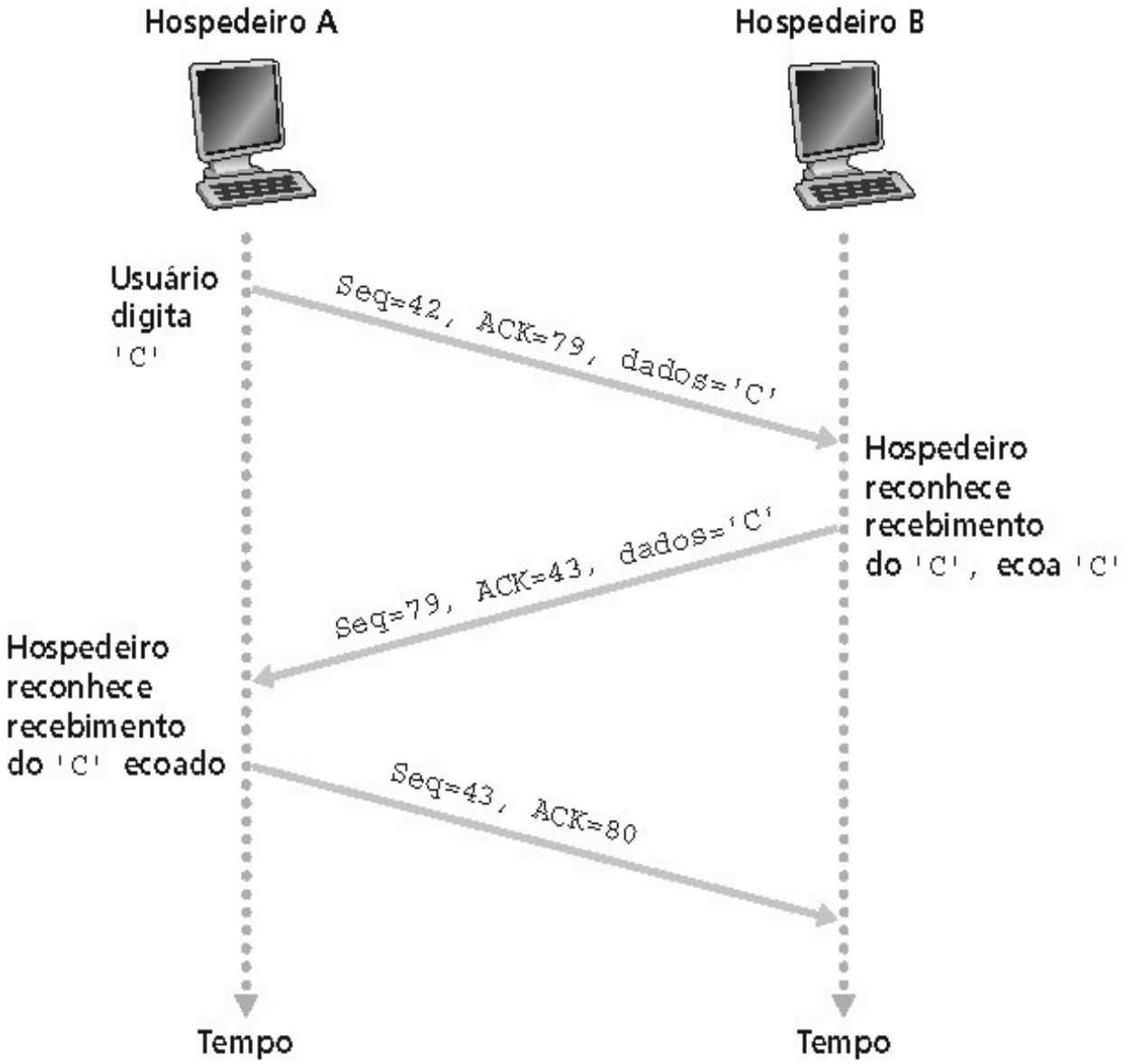


Estados do servidor

Números de seqüência e ACK

- Números de seqüência:
  - Número do primeiro byte nos segmentos de dados.
- ACKs:
  - Número do próximo byte esperado do outro lado;
  - ACK cumulativo.
- Como o receptor trata segmentos fora de ordem?
  - A especificação do TCP não define, fica a critério do implementador.

Estudo de Caso: Telnet



Como determinar o valor dos temporizadores?

Como determinar o valor dos temporizadores?

- Maior que o RTT (Round Trip Time)!
  - Porém RTT varia!
- Muito curto: temporização prematura.
  - Retransmissões desnecessárias.
- Muito longo: a reação à perda de segmento fica lenta.
- Como estimar o RTT?

Amostras de RTT

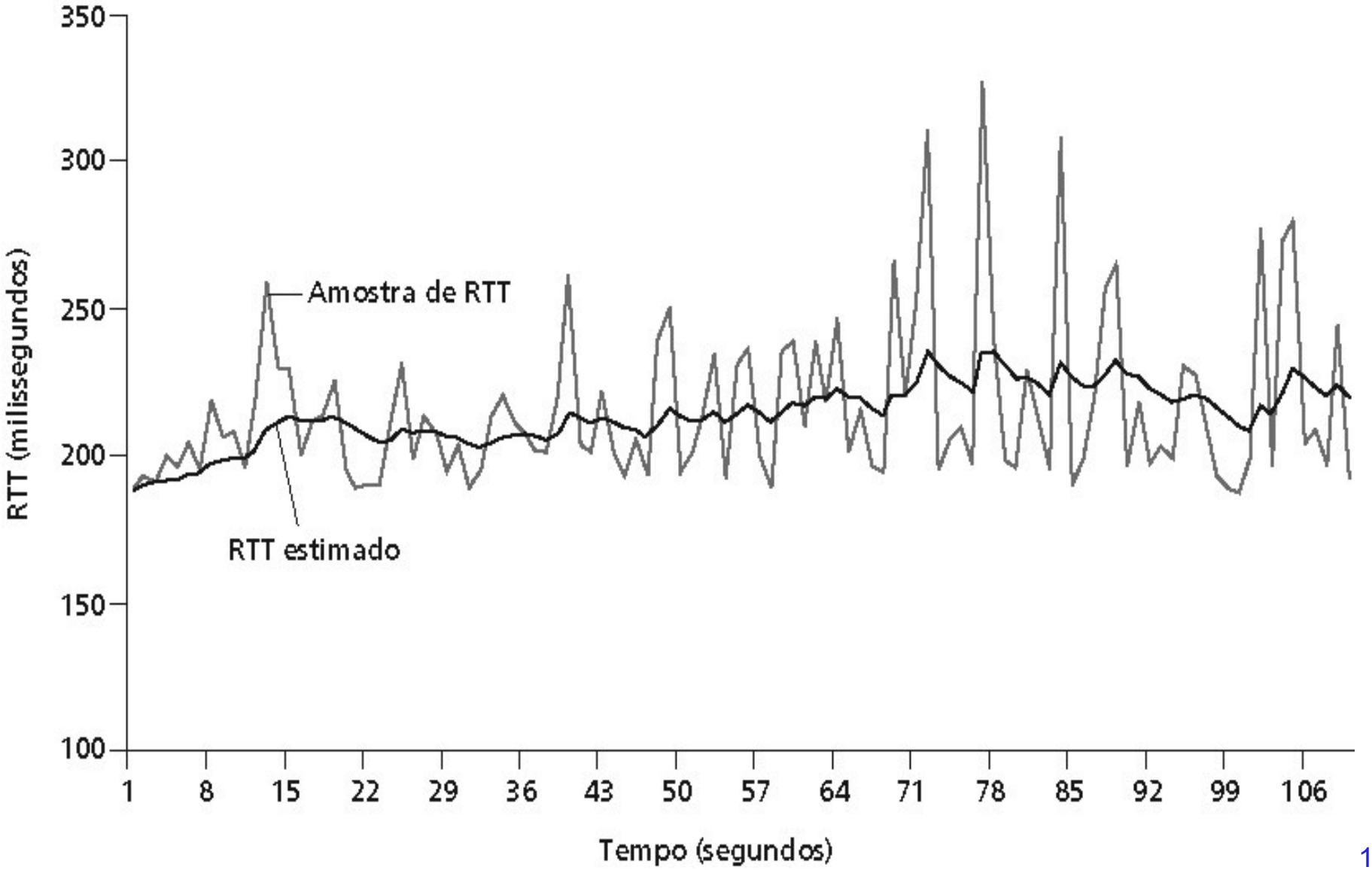
- **SampleRTT**: tempo medido da transmissão de um segmento até a respectiva confirmação.
  - Ignora retransmissões e segmentos reconhecidos de forma cumulativa.
- SampleRTT varia de forma rápida, é desejável um amortecedor para a estimativa do RTT.
  - Usar várias medidas recentes, não apenas o último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

Estimando o RTT

EstimatedRTT = (1-α)*EstimatedRTT + α*SampleRTT

- Média móvel com peso exponencial
- Influência de uma dada amostra decresce de forma exponencial
- Valor típico: $\alpha = 0{,}125$

Exemplo de Estimativa do RTT



Definindo a temporização

- **EstimatedRTT** mais "margem de segurança".
- Primeiro estimar o quanto o SampleRTT se desvia do EstimatedRTT:

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

(tipicamente $\beta = 0.25$)

Definindo a temporização

- Então ajustar o intervalo de temporização:

`TimeoutInterval = EstimatedRTT + 4*DevRTT`

TCP: Transferência de Dados Confiável

TCP: Transferência de Dados Confiável

- TCP cria serviços de rdt em cima do serviço não-confiável do IP.
  - segmentos em paralelo;
  - ACKs cumulativos;
  - TCP usa tempo de retransmissão simples.
- Retransmissões são disparadas por:
  - eventos de tempo de confirmação;
  - ACKs duplicados.

Eventos do Transmissor TCP

- Dado recebido da aplicação:
  - cria segmento com número de sequência;
  - Número de sequência é o número do byte-stream do 1º byte de dados no segmento;
  - inicia o temporizador se ele ainda não estiver em execução (considera o segmento não-confirmado mais antigo);
  - tempo de expiração: TimeOutInterval.

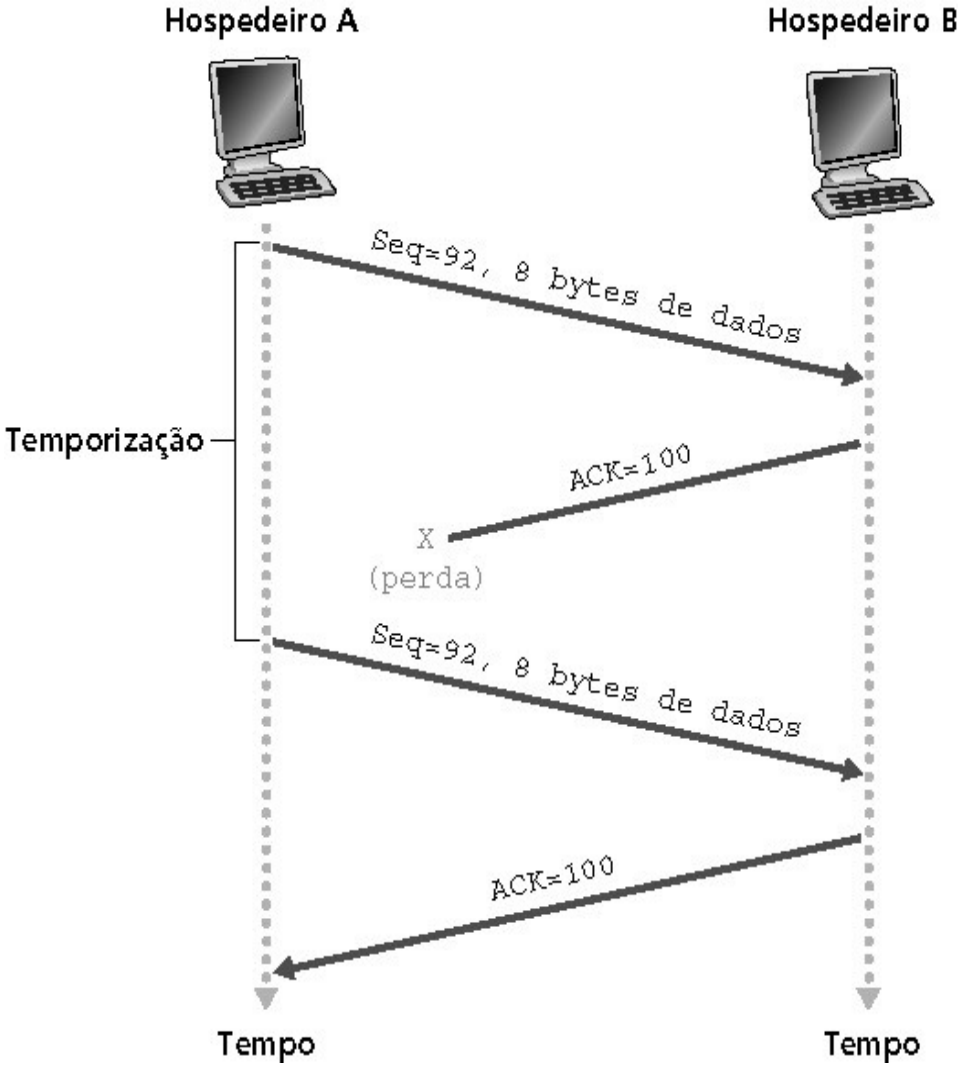
Eventos do Transmissor TCP

- Tempo de confirmação (timeout):
  - 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.

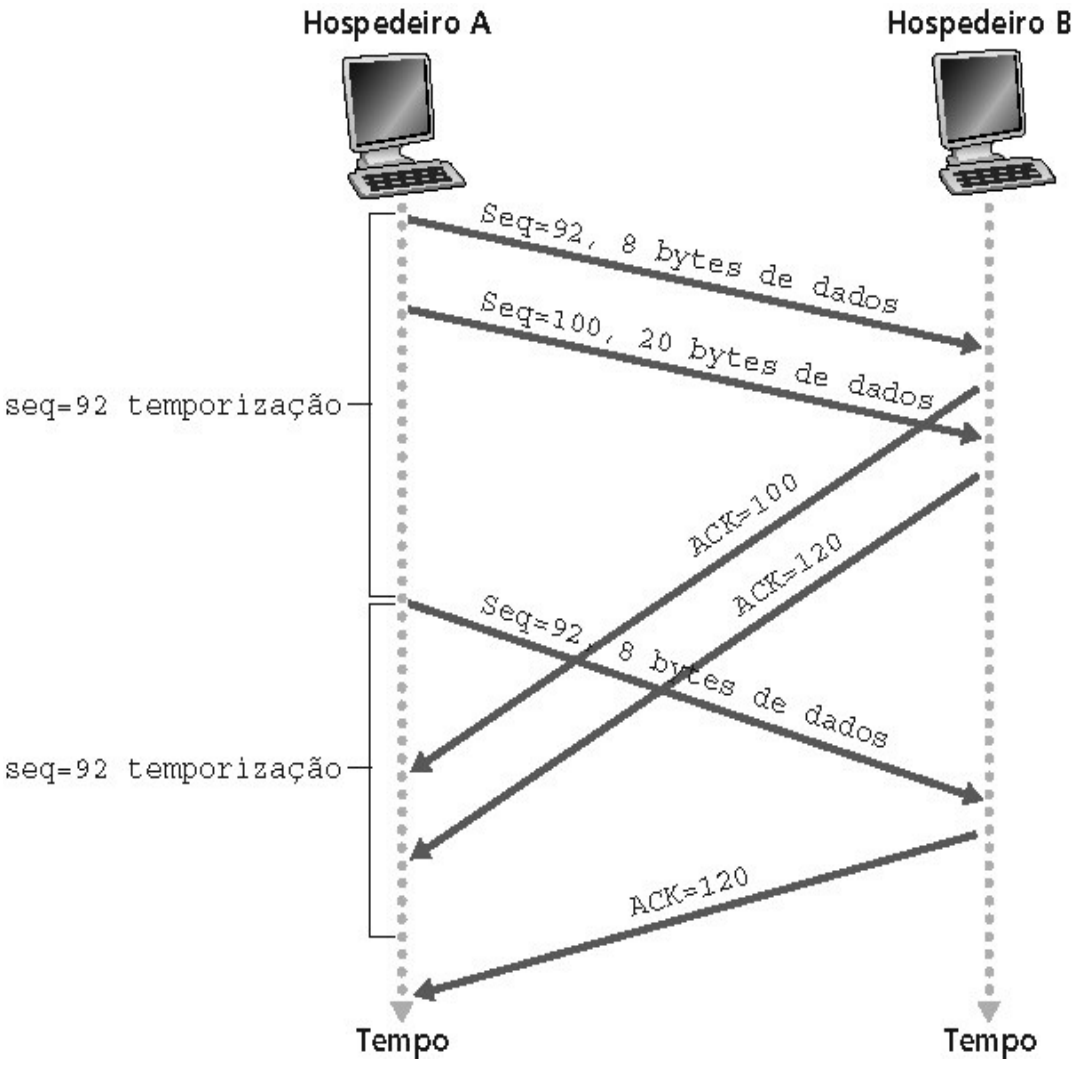
```
/* Assume sender is not constrained by TCP flow or congestion con-
trol, that data from above is less than MSS in size, and that data
transfer is in one direction only. */

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
   switch(event)

      event: data received from application above
         create TCP segment with sequence number NextSeqNum
         if (timer currently not running)
            start timer
         pass segment to IP
         NextSeqNum=NextSeqNum+length(data)
         break;

      event: timer timeout
         retransmit not-yet-acknowledged segment with
            smallest sequence number
         start timer
         break;

      event: ACK received, with ACK field value of y
         if (y > SendBase) {
            SendBase=y
            if (there are currently any not-yet-acknowledged segments)
               start timer
            }
            break;

   } /* end of loop forever */
```

Figure 3.33 ♦ Simplified TCP sender

TCP: Retransmissão

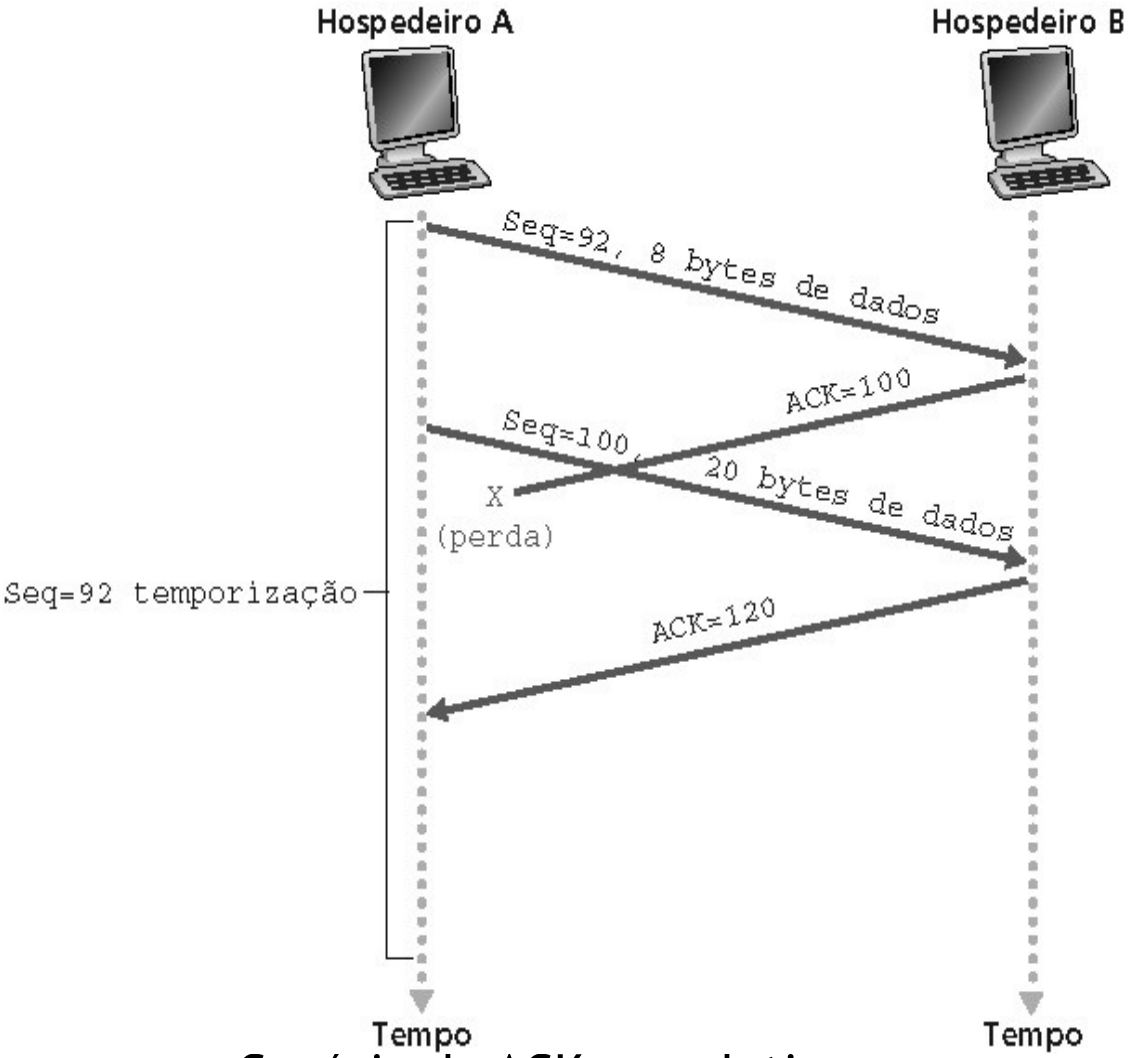


Cenário com perda do ACK



Temporização prematura, ACKs cumulativos

TCP: Retransmissão



Cenário de ACK cumulativo

Geração de ACK [RFC 1122,2581]

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.

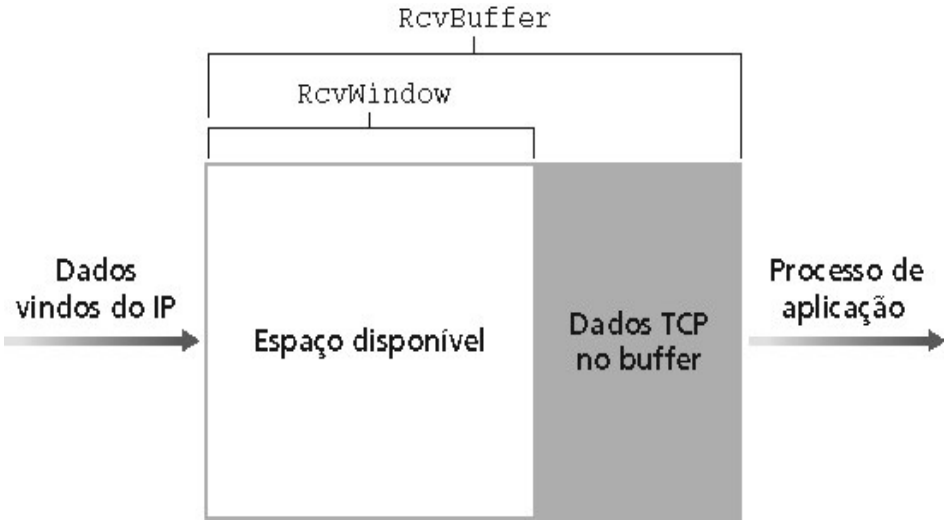
Retransmissão Rápida

- Com frequência, o tempo de expiração é relativamente longo...
- Detecta segmentos perdidos por meio de ACKs duplicados:
  - transmissor frequentemente envia muitos segmentos *back-to-back,* e se um segmento é perdido, haverá muitos ACKs duplicados.
- Se o transmissor recebe 3 ACKs para o mesmo dado, supõe que o segmento após foi perdido:
  - Retransmissão rápida: reenvia o segmento antes de o temporizador expirar.

TCP: Controle de fluxo

Controle de Fluxo do TCP

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



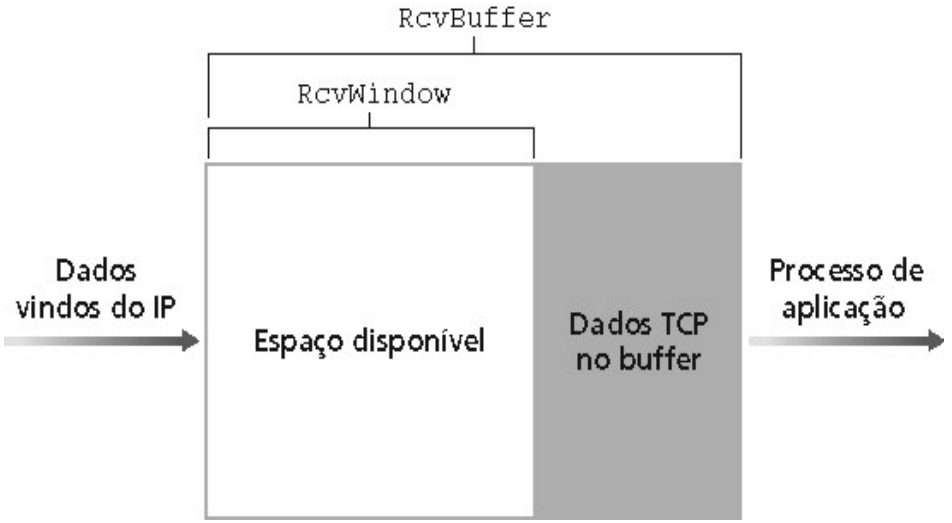
- Processos de aplicação podem ser lentos para ler o buffer

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.

Controle de Fluxo do TCP: Como funciona?



- Receptor informa a área disponível 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]`

TCP: 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).

Mecanismos de Controle de Congestionamento

Fim-a-fim:

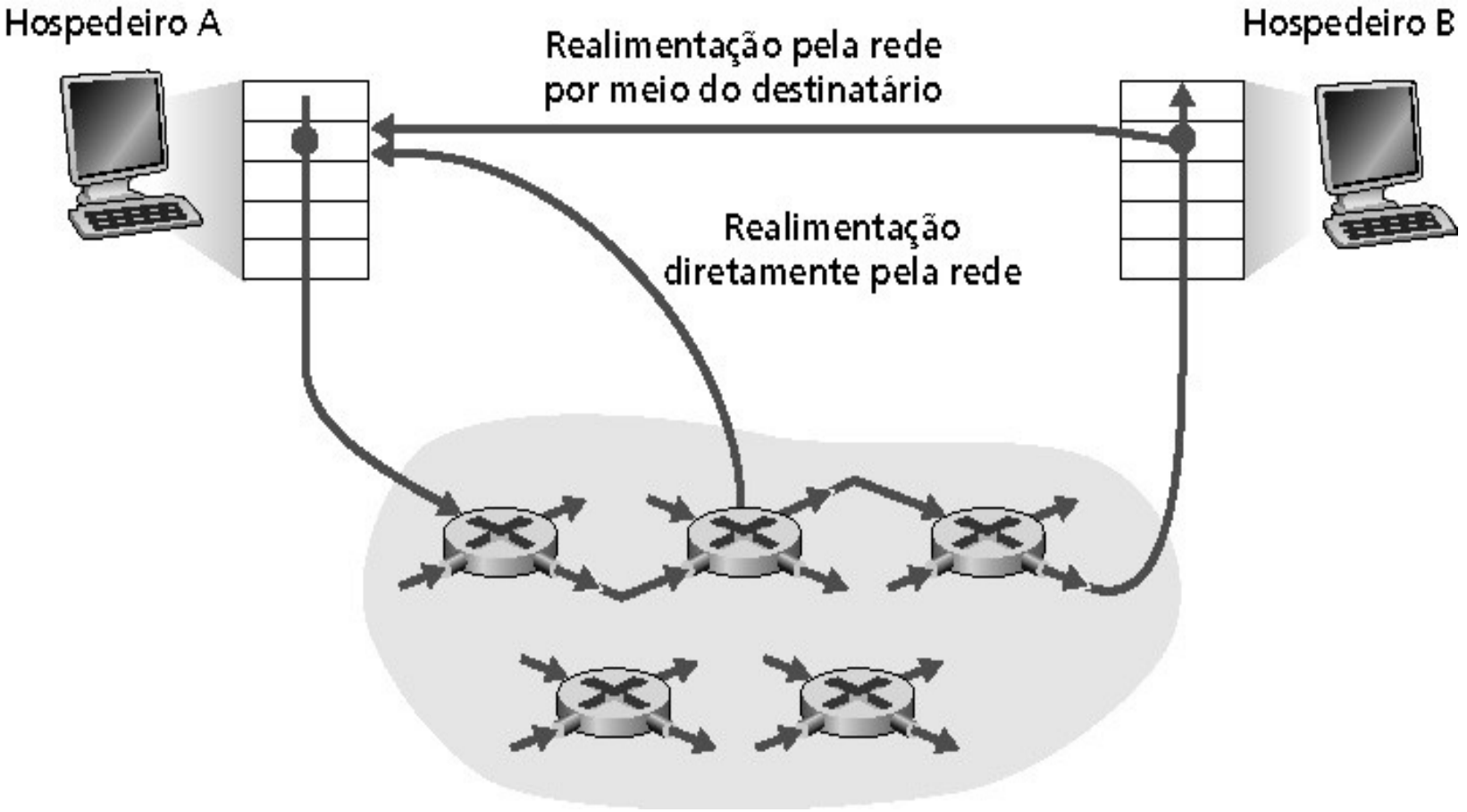
- Não usa realimentação explícita da rede.
- Congestionamento é inferido a partir das perdas e dos atrasos observados nos sistemas finais.
- Abordagem usada pelo TCP.

Mecanismos de 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.

Mecanismos de Controle de Congestionamento



TCP: Controle de Congestionamento

TCP: Controle de Congestionamento

- Controle fim-a-fim (sem assistência da rede).
- Transmissor limita a transmissão:
  LastByteSent - LastByteAcked ≤ CongWin
- Aproximadamente,
  rate = CongWin / RTT Bytes/seg
- **CongWin** é dinâmico, função de congestionamento das redes detectadas.

Como o transmissor detecta congestionamento?

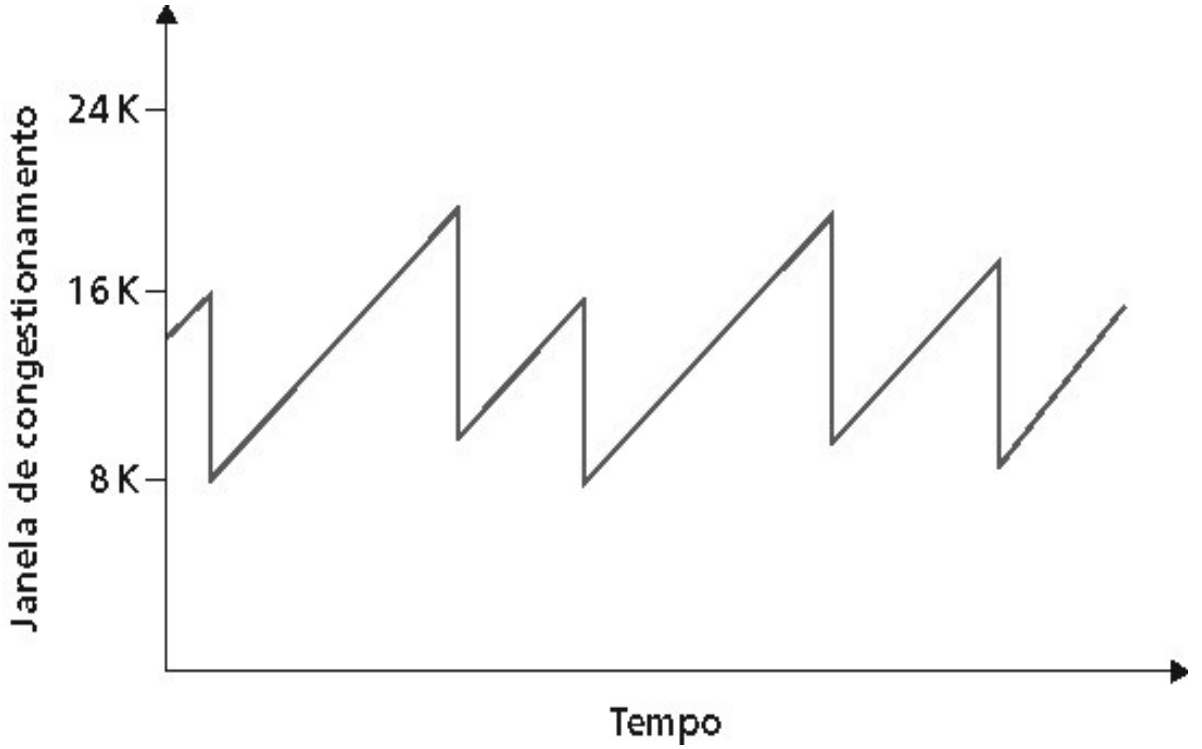
- Através da perda de segmentos!
- A perda é detectada por:
  - tempo de confirmação (timeout);
  - *ou* 3 ACKs duplicados.
- Consequentemente, transmissor TCP reduz a taxa (**CongWin**) após o evento de perda.

TCP: Controle de Congestionamento

- Algoritmo possui três mecanismos:
  - Aumento Aditivio, Diminuição Multiplicativa (AIMD);
  - Partida lenta;
  - Reação a eventos de esgotamento de temporização.

TCP: AIMD (additive increase/multiplicative decrease)

- **Redução multiplicativa**: diminui o **CongWin** pela metade após o evento de perda.
- **Aumento aditivo**: aumenta o **CongWin** com 1 MSS a cada RTT na ausência de eventos de perda.



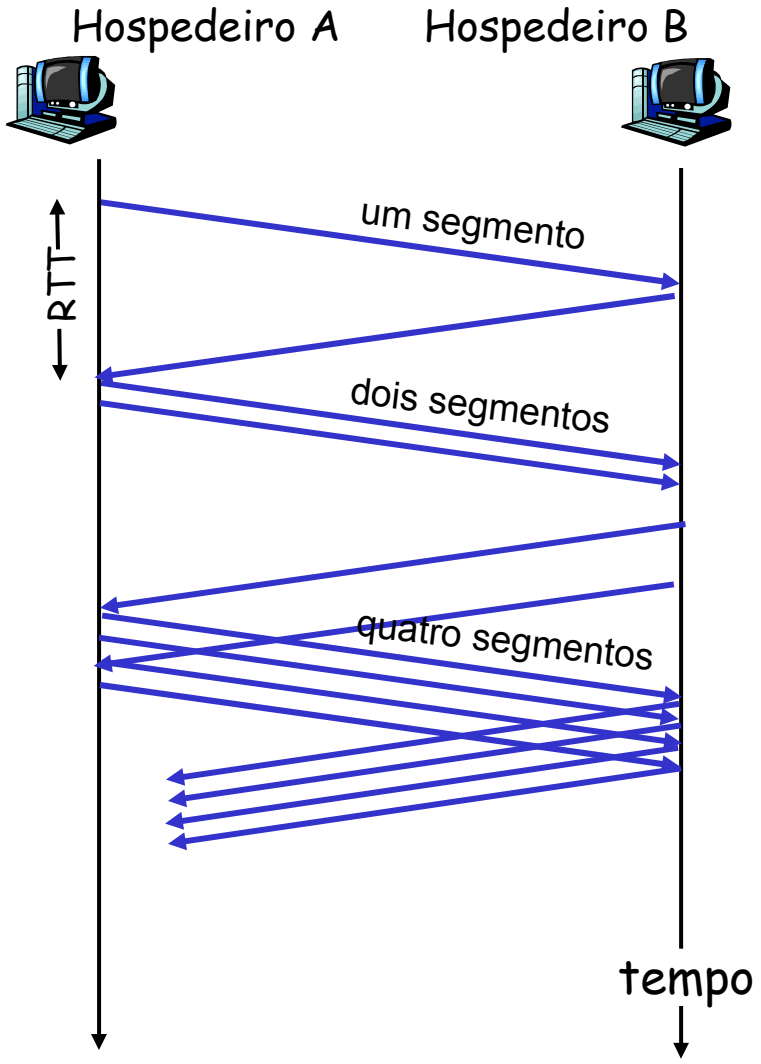
TCP: Partida Lenta

- Quando a conexão começa, **CongWin** = 1 MSS
  - Exemplo: MSS = 500 bytes e RTT = 200 ms.
  - Taxa inicial = 20 kbps.
- Largura de banda disponível pode ser >> MSS/RTT
  - Desejável aumentar rapidamente até a taxa respeitável.

TCP: Partida Lenta

- Quando a conexão começa, a taxa aumenta 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.

Resumindo: taxa inicial é lenta, mas aumenta de modo exponencialmente rápido.



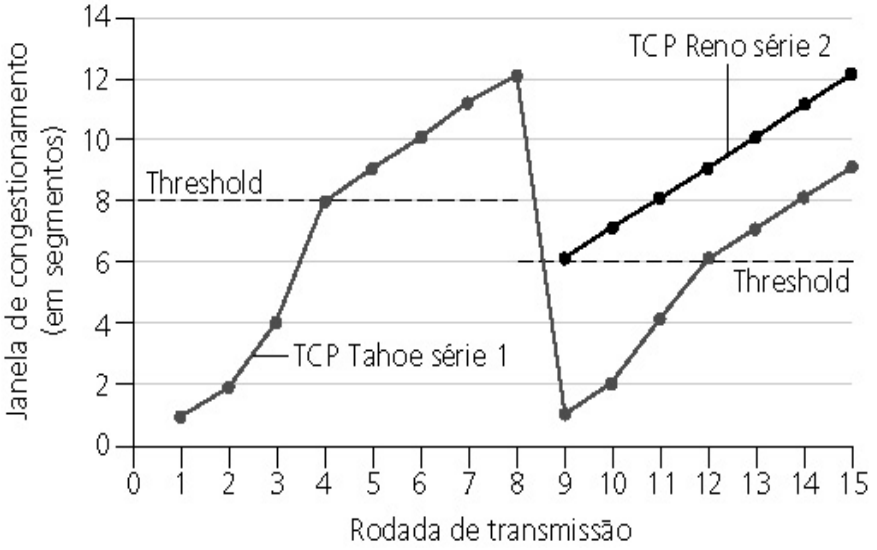
TCP Reno: Reação a *timeouts*

- ∀ Após 3 ACKs duplicados:
  - **`CongWin`** é cortado pela metade;
  - janela então cresce linearmente.
- ∀ Após *timeout* de confirmação:
  - **`CongWin`** é ajustado para 1 MSS;
  - janela então cresce exponencialmente até um limite, então cresce linearmente.

- **Filosofia:**
  - se ocorreu *timeout*, dados não estão sendo entregues!

TCP: Reação a *timeouts*

- Quando o aumento exponencial deve tornar-se linear?
- Quando **CongWin** obtiver 1/2 do seu valor antes do tempo de confirmação.
- Implementação:
  - No evento de perda, o limiar (variável) é ajustado para 1/2 do CongWin logo antes do evento de perda.

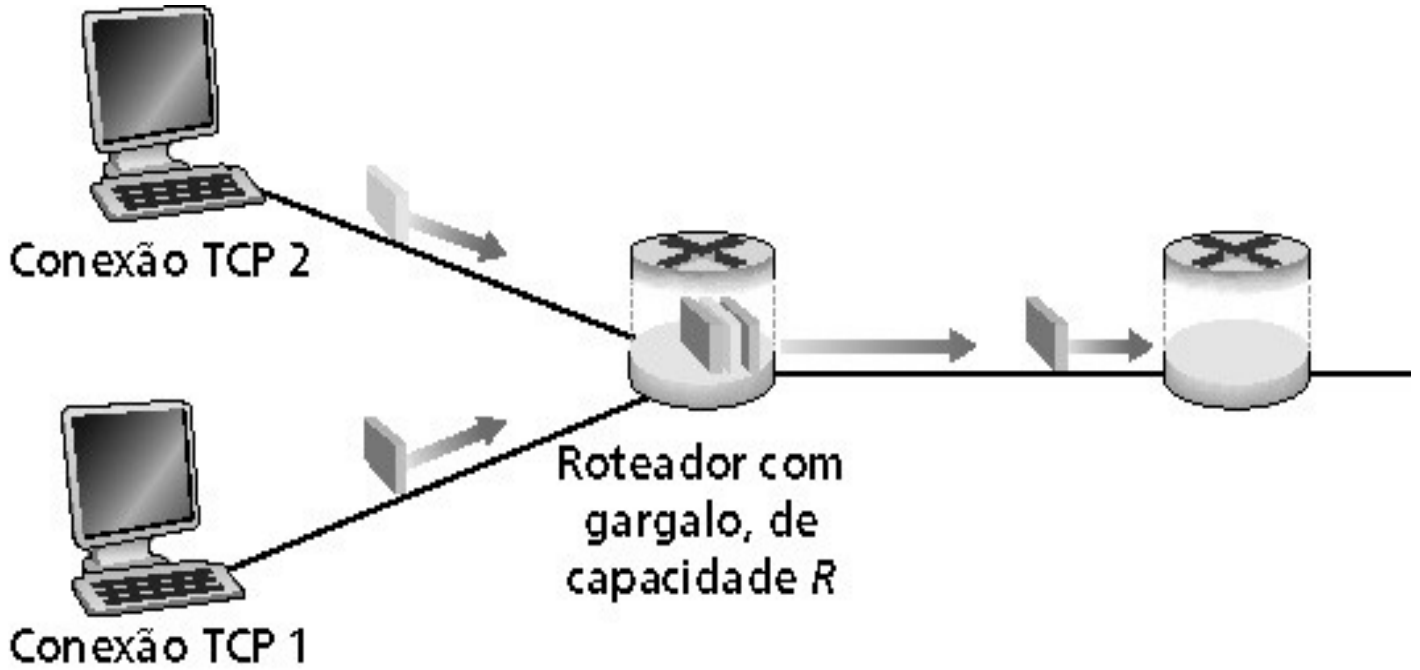


TCP: Controle de Congestionamento

- 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 (timeout), o **Threshold** é ajustado para **CongWin/2** e o **CongWin** é ajustado para 1 MSS.

Justiça (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.



Justiça 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

Justiça e Conexões TCP Paralelas

- Nada previne as aplicações de abrirem conexões paralelas entre 2 hospedeiros!
- 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 (uma única aplicação com a metade da banda por criar conexões paralelas)

Modelagem do atraso TCP

- Quanto tempo demora para receber um objeto de um servidor Web após enviar um pedido? Ou seja, qual a latência?
- Ignorando o congestionamento, o atraso é influenciado por:
  - estabelecimento de conexão TCP;
  - atraso de transferência de dados;
  - partida lenta.

Modelagem do atraso TCP - Hipóteses

- Suponha um enlace entre o cliente e o servidor com taxa de dados R.
- Seja:
  - S: MSS (bits);
  - O: tamanho do objeto (bits).
- Não há retransmissões (sem perdas e corrupção de dados).
- Janela de congestionamento (W):
  - estática;
  - dinâmica.

Janela de Congestionamento Estática

Primeiro caso:

- WS/R > RTT + S/R
- ou seja, o ACK para o primeiro segmento na primeira janela retorna antes de enviar todos os dados.

atraso = 2RTT + O/R

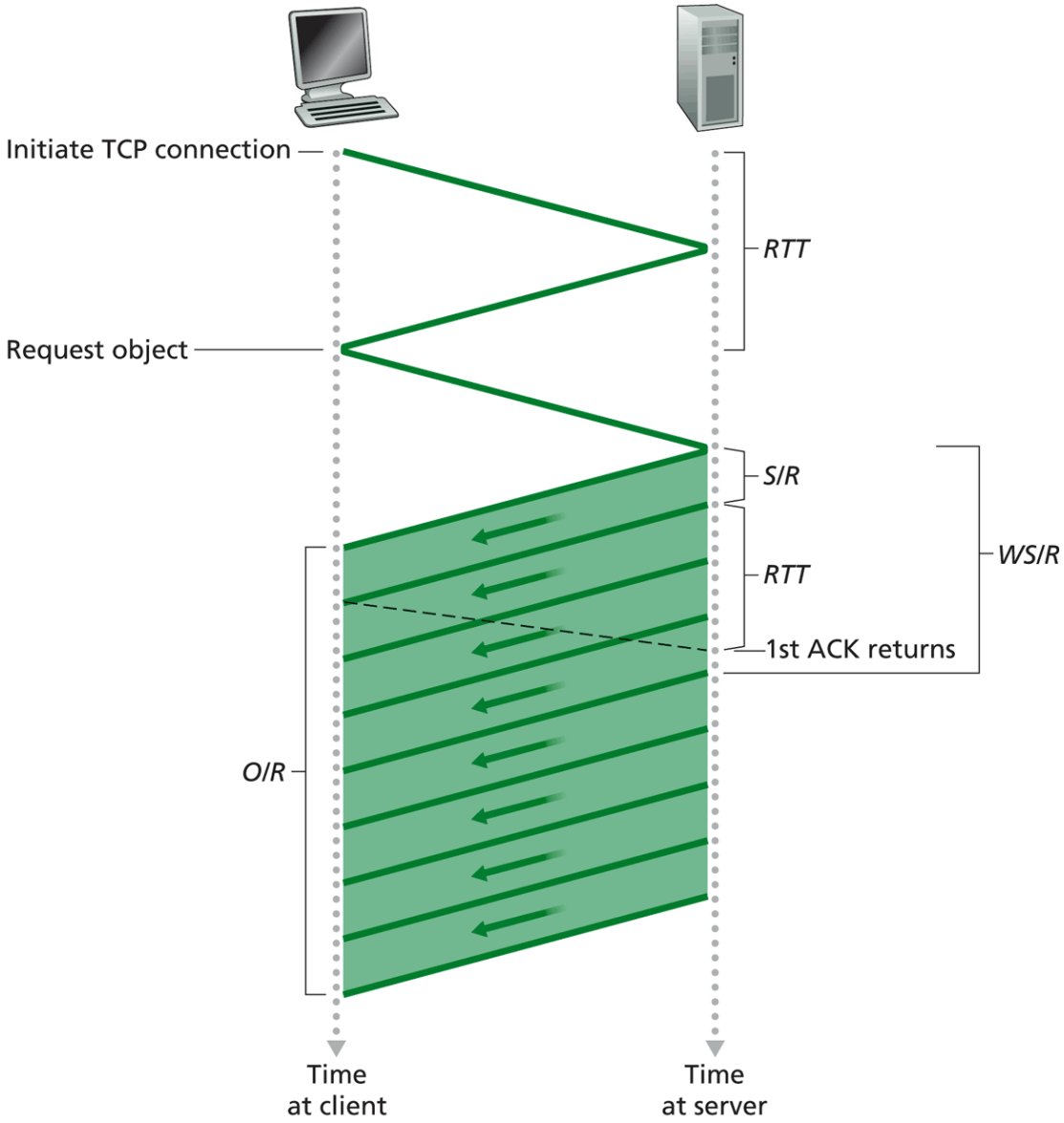


Figure 3.55 ♦ The case $WS/R > RTT + S/R$

Janela de Congestionamento Estática

Segundo caso:

- ∀ WS/R < RTT + S/R
- ∀ ou seja, o ACK chega após enviar a janela de dados completa.
- ∀ K = O/WS (No. de janelas de dados q abrange o objeto

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

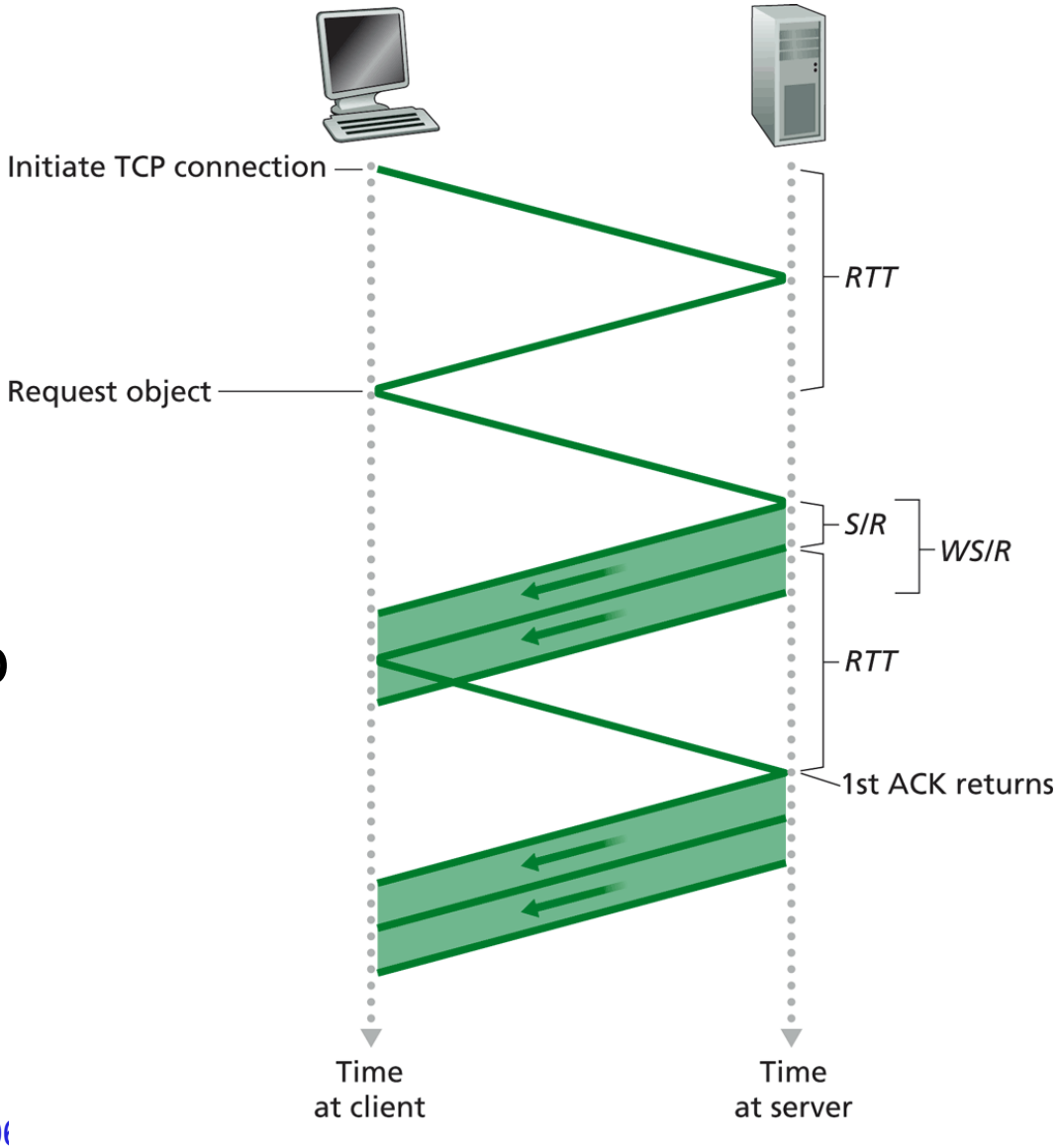


Figure 3.56 ♦ The case *WS/R < RTT + S/R*

Janela Dinâmica

Agora suponha que a janela cresca de acordo com os procedimentos da fase partida lenta.

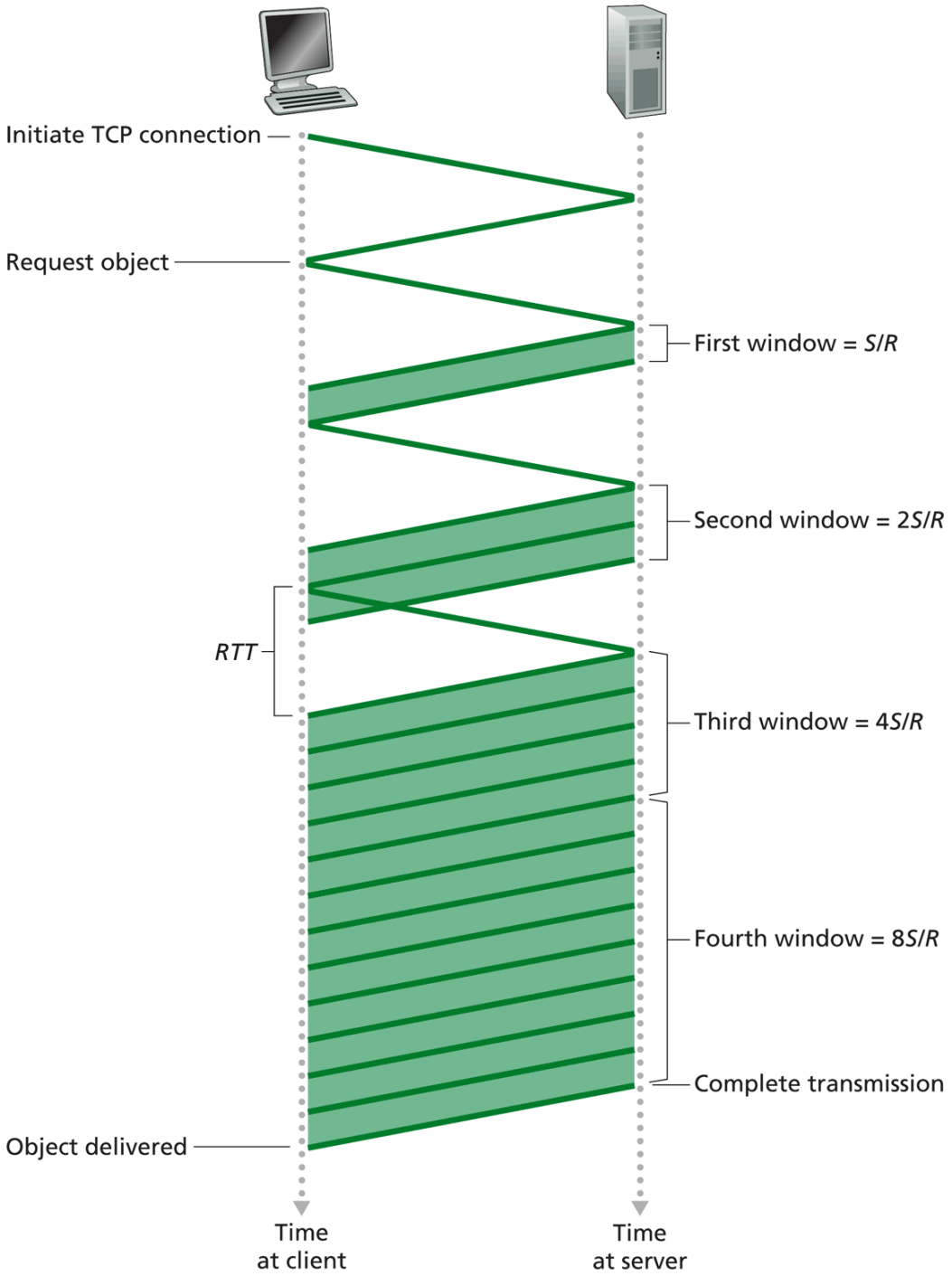


Figure 3.57 ♦ TCP timing during slow start

Janela de Congestionamento Dinâmica - Atraso

Componentes do atraso:

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

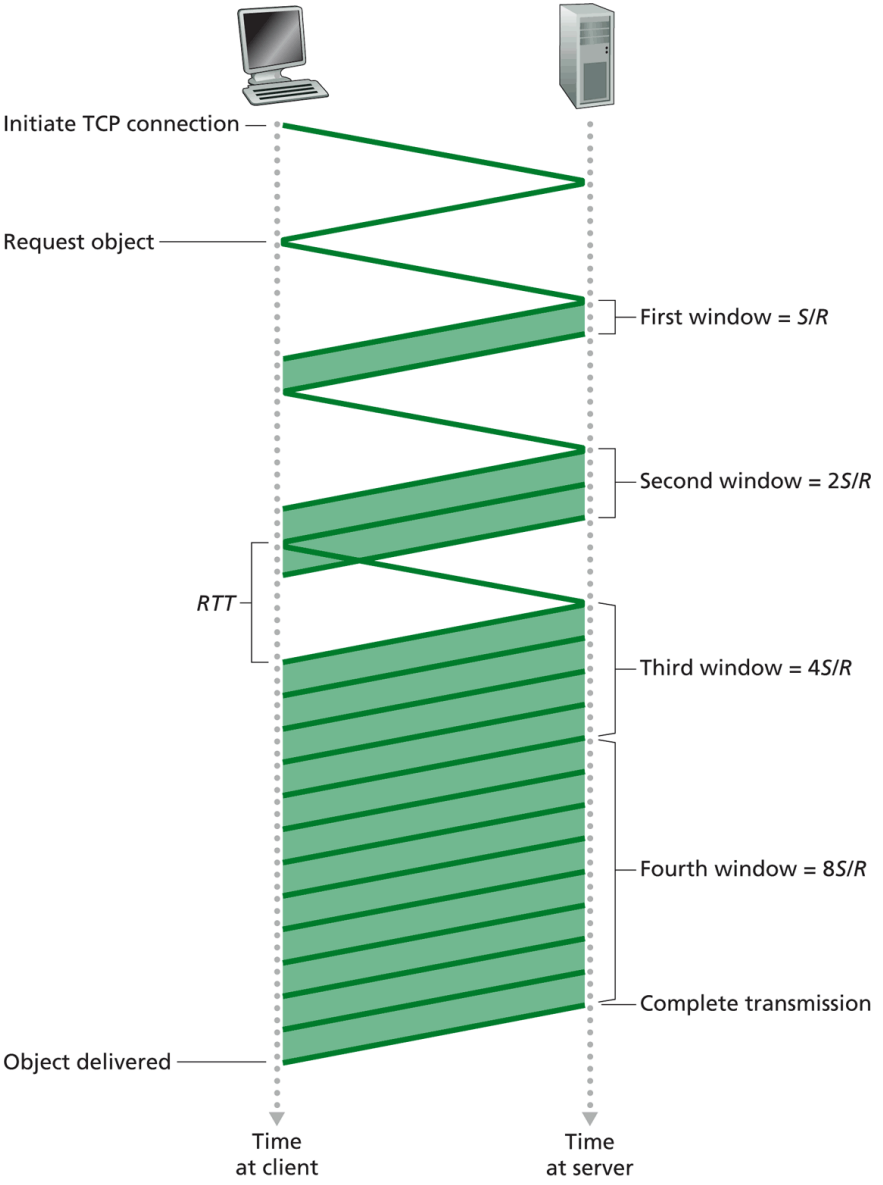


Figure 3.57 ♦ TCP timing during slow start

Modelagem de atraso: HTTP

Assuma 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**

A seguir...

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