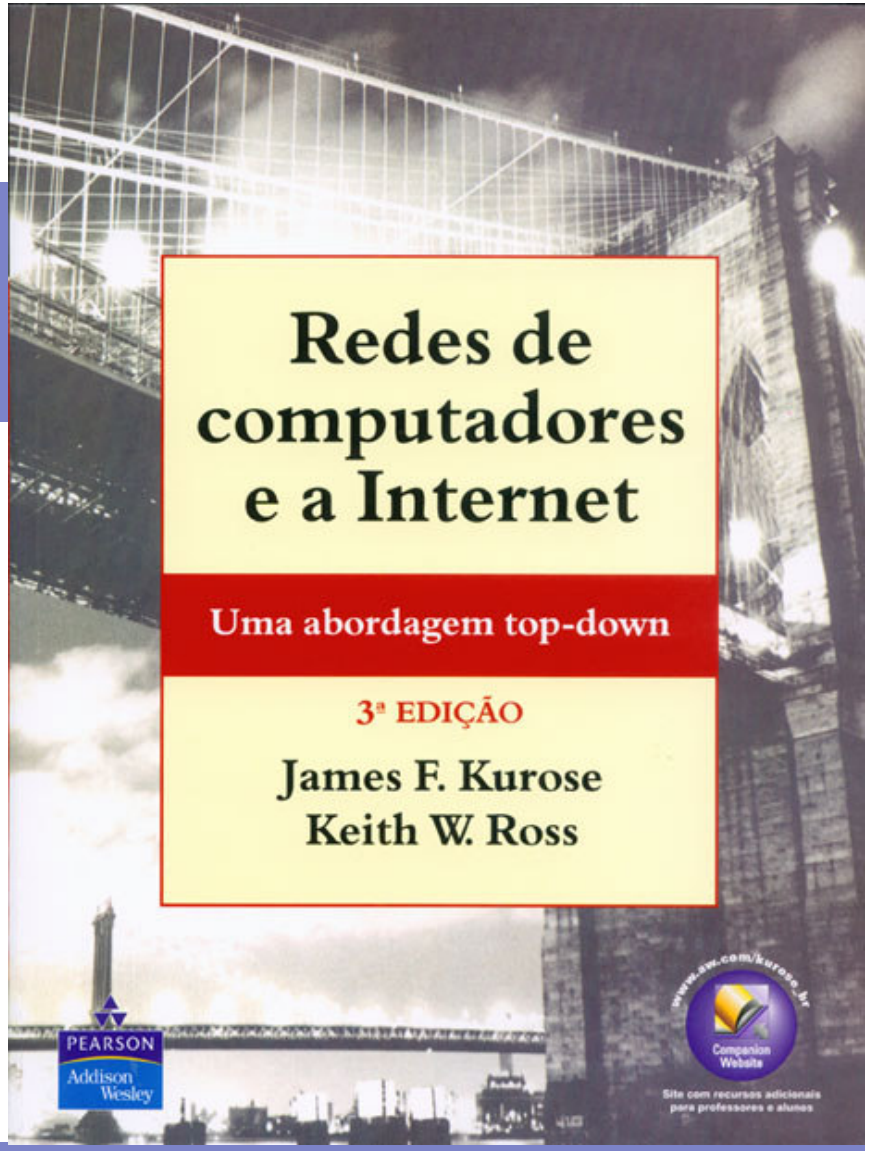
Redes de computadores e a Internet

Prof. Gustavo Wagner

Capítulo 3

Camada de transporte



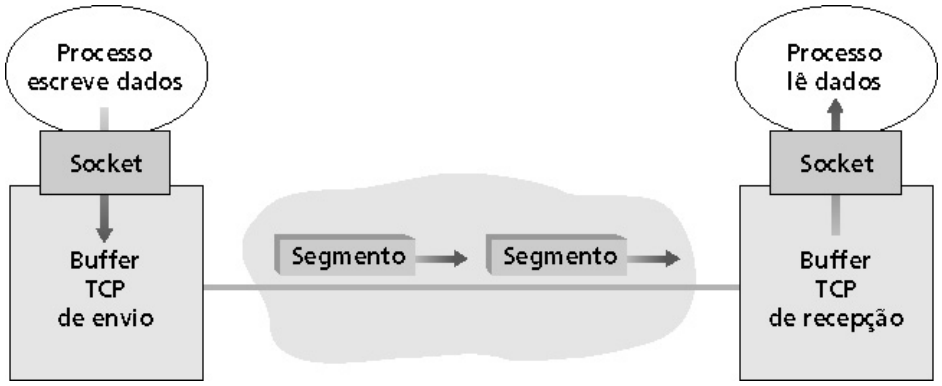
3 Camada de transporte

- 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

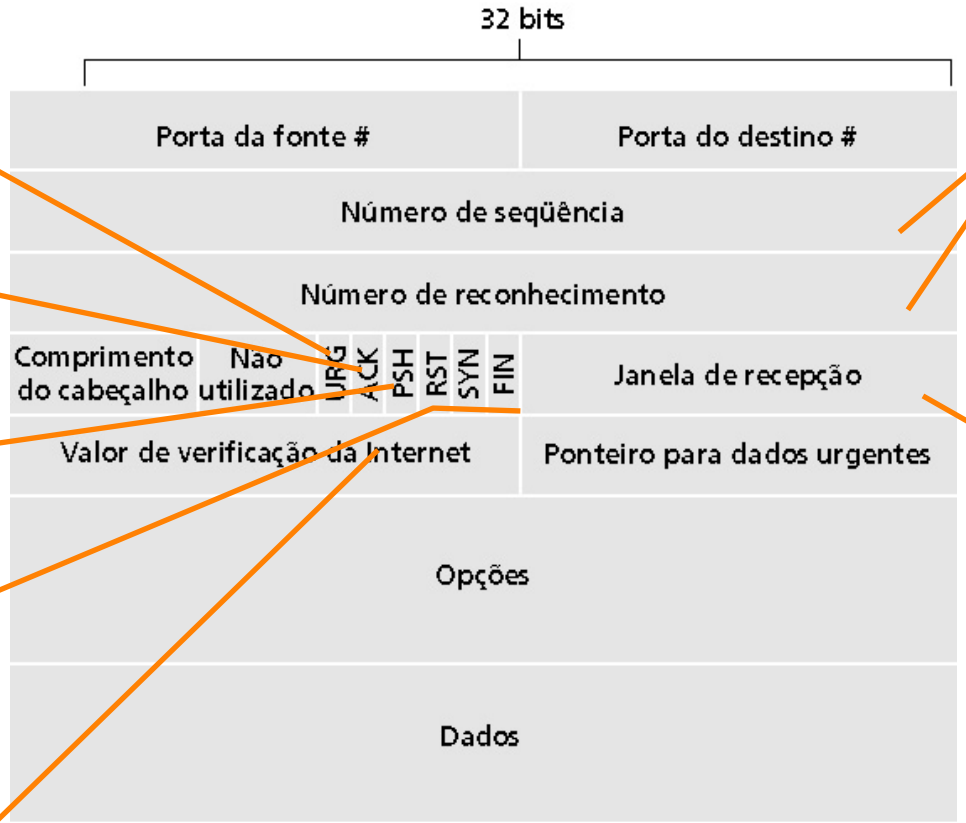
3 TCP: overview

RFCs: 793, 1122, 1323, 2018, 2581

- **Ponto-a-ponto:**
  - Um transmissor, um receptor
- **Confiável, seqüencial byte stream:**
  - Não há contornos de mensagens
- **Pipelined:** (transmissão de vários pacotes sem confirmação)
  - Controle de congestão e de fluxo definem tamanho da janela
- **Buffers de transmissão e de recepção**
- Dados full-duplex:
  - Transmissão bidirecional na mesma conexão
  - MSS: maximum segment size
- Orientado à conexão:
  - Apresentação (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
  - Applet



3 Estrutura do segmento TCP



URG: dados urgentes (pouco usados)

ACK: campo de ACK é válido

PSH: produz envio de dados (pouco usado)

RST, SYN, FIN: estabelec. de 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

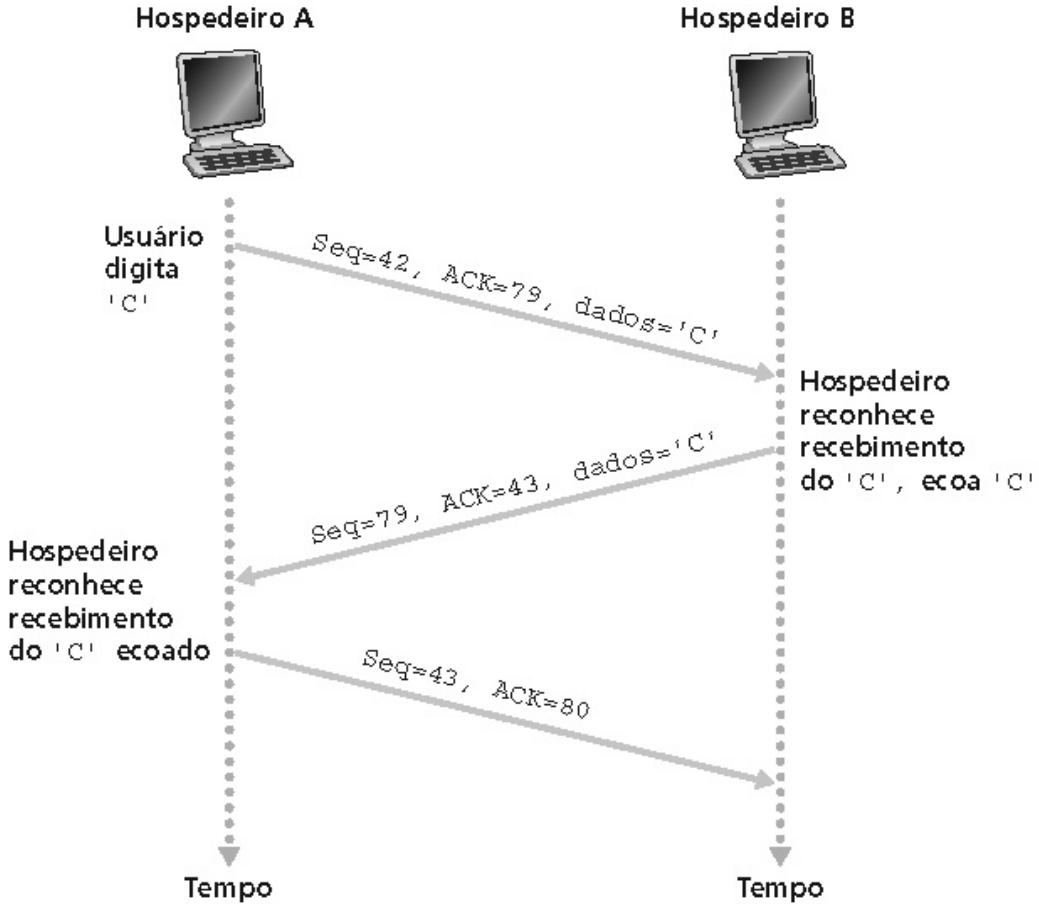
3 Número de seqüência e ACKs do TCP

Números de seqüência:
- Número do primeiro byte nos segmentos de dados

ACKs:
- Número do próximo byte esperado do outro lado
- ACK cumulativo

P.: Como o receptor trata segmentos fora de ordem?

- A especificação do TCP não define, fica a critério do implementador
- Duas alternativas:
  - Destinatário descarta o segmento;
  - Destinatário conserva bytes for a de ordem (essa opção é a adotada na prática);



Exemplo de Telnet

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

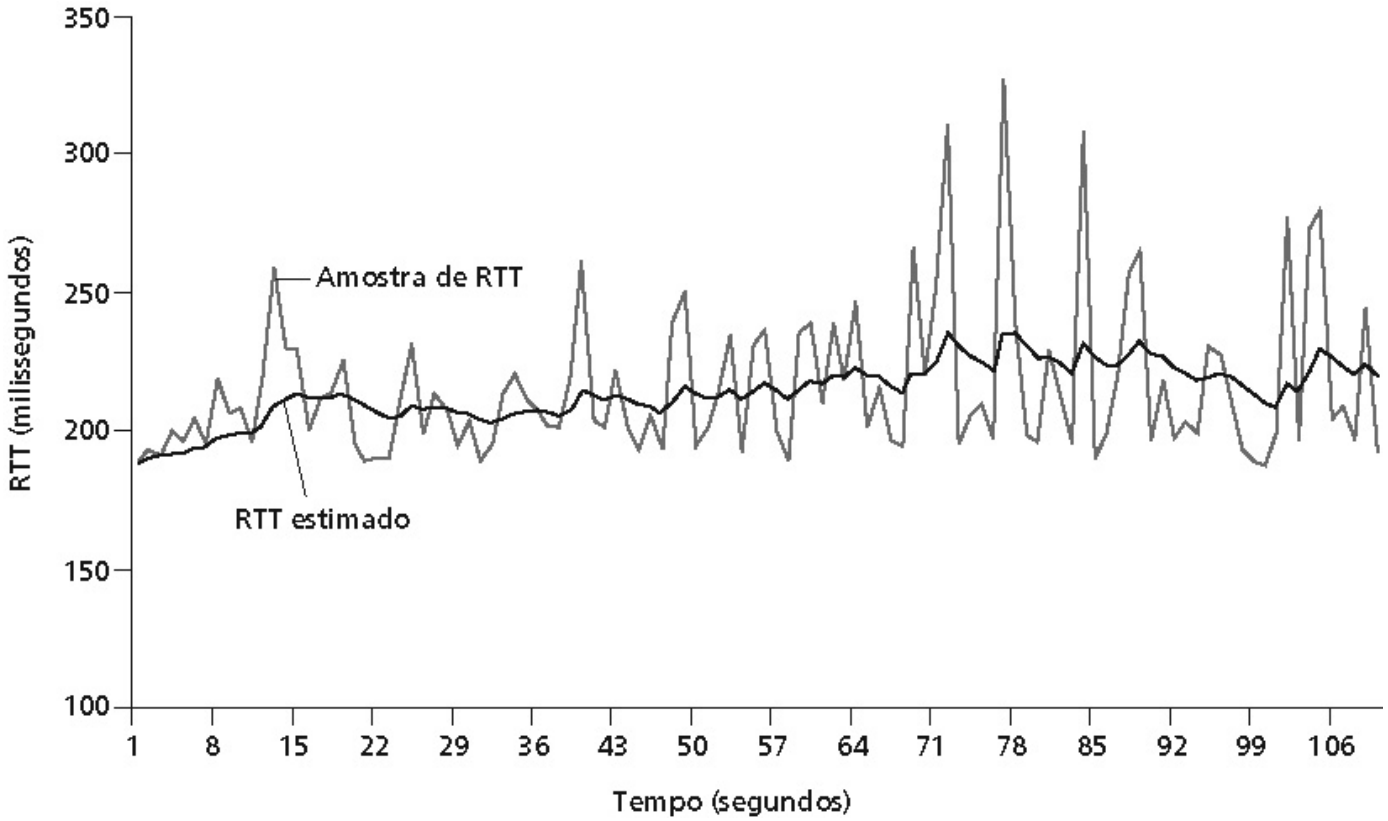
3 TCP Round Trip Time e temporização

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

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

$$\mathbf{EstimatedRTT = 0,875 * EstimatedRTT + 0.125 * SampleRTT}$$

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 */
```

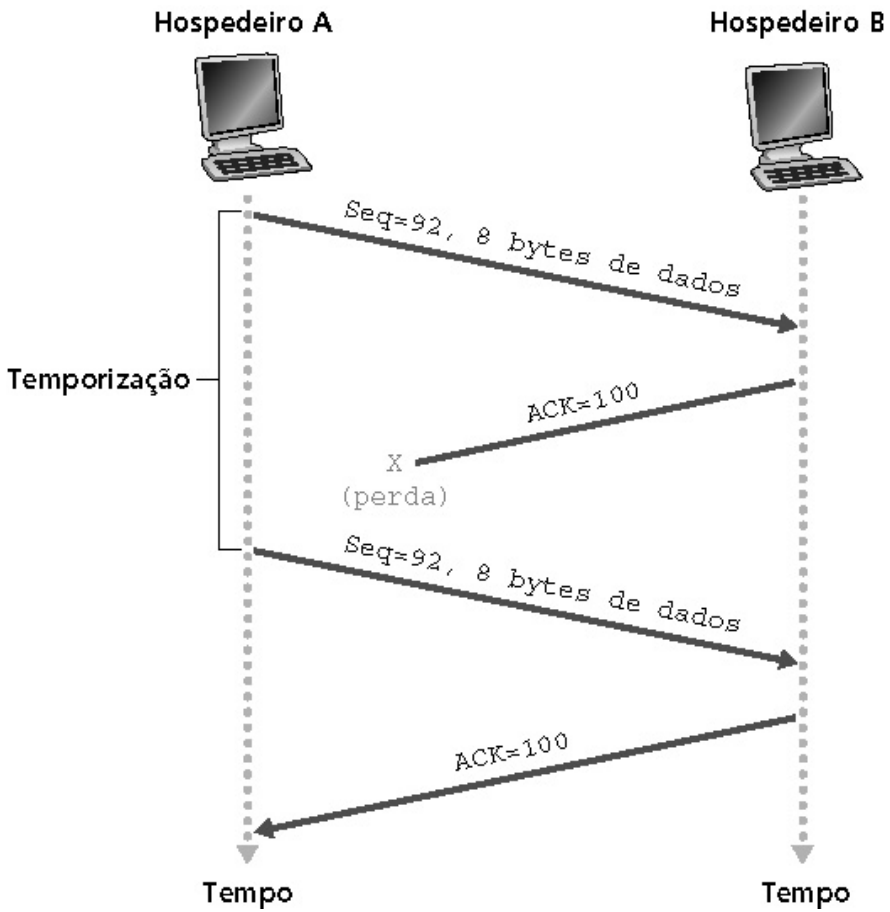
Comentário:

- SendBase-1: último byte do ACK cumulativo

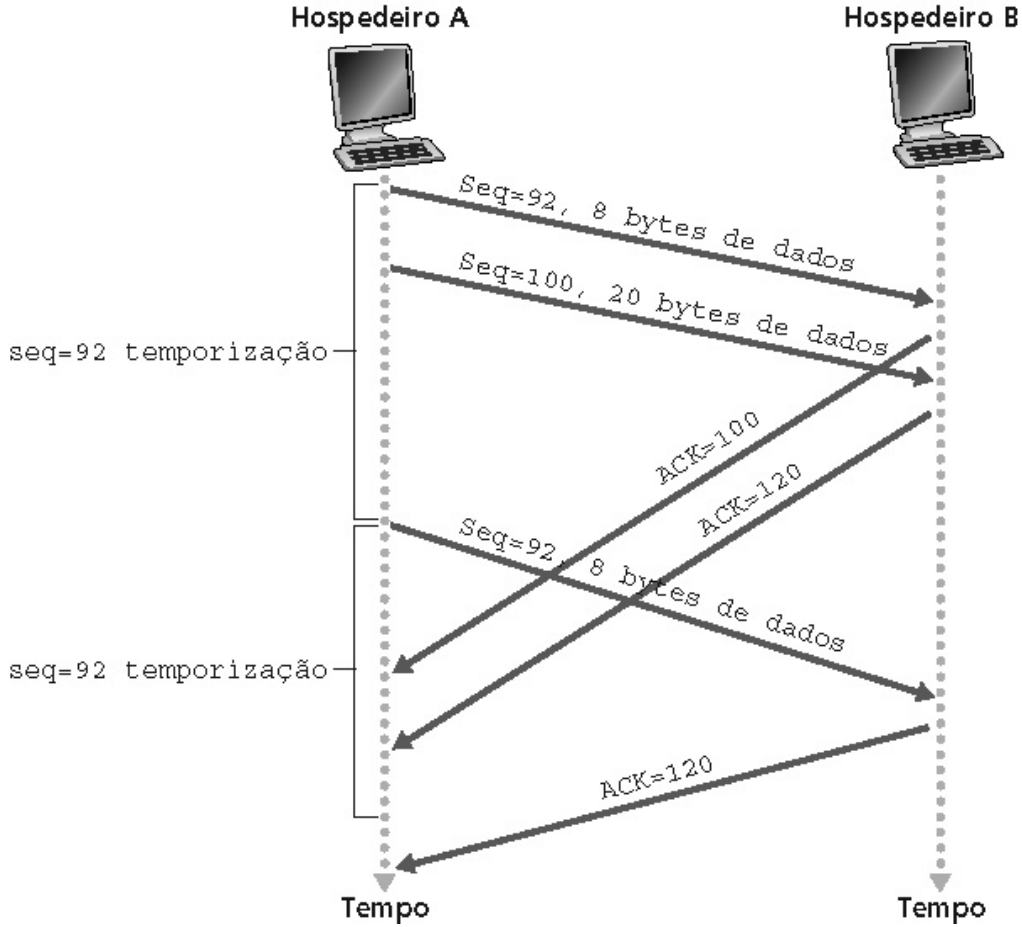
Exemplo:

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

3 TCP: cenários de retransmissão

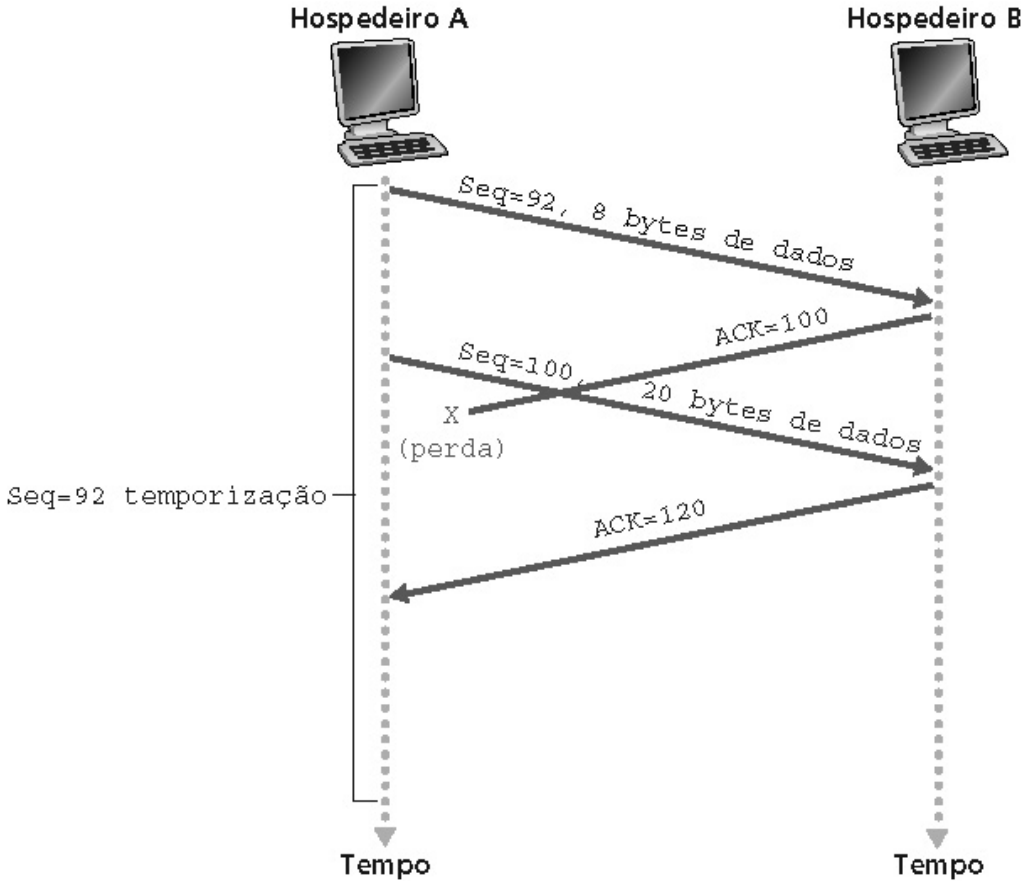


Cenário com perda do ACK



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

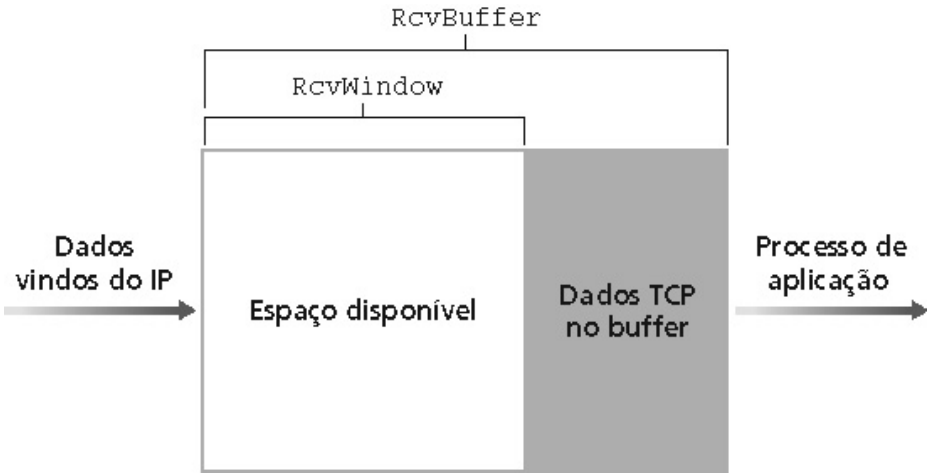
retransmissão rápida

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:



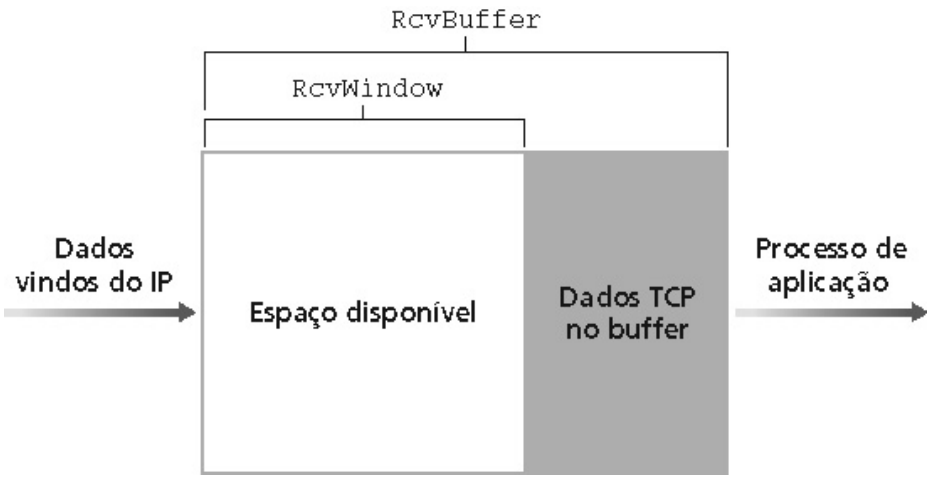
- Processos de aplicação podem ser lentos para ler o buffer

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 disponível incluindo valor **RcvWindow** nos segmentos
- Transmissor limita os dados não confimados 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]

Applet

• E se RcvWindow=0, quando o Transmissor voltará a reenviar, já que o receptor não envia mais pacotes para atualizar o estado do RcvWindow?

• Transmissor enviar um pacote com um único byte, só para saber o valor do RcvWindow;

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

Passo 3: o sistema final cliente reconhece o SYNACK

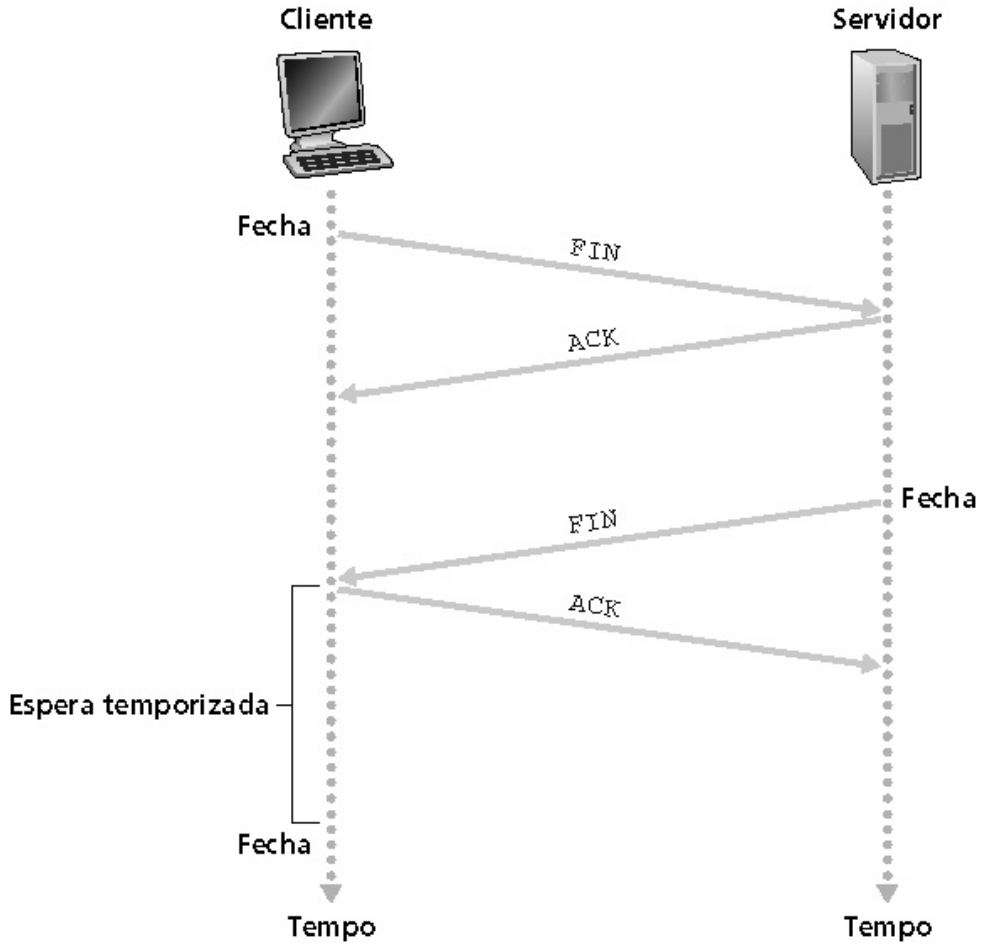
3 Gerenciamento de conexão TCP

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



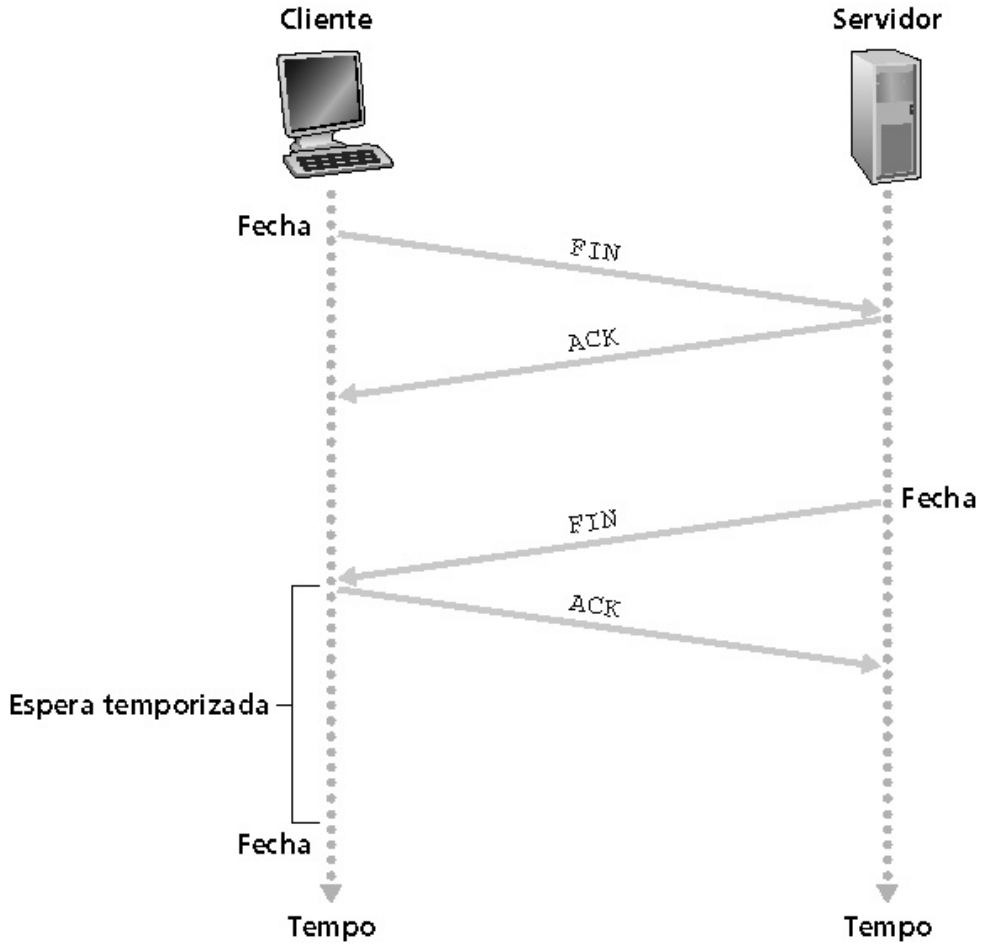
3 Gerenciamento de conexão TCP

Passo 3: cliente recebe FIN, responde com ACK.

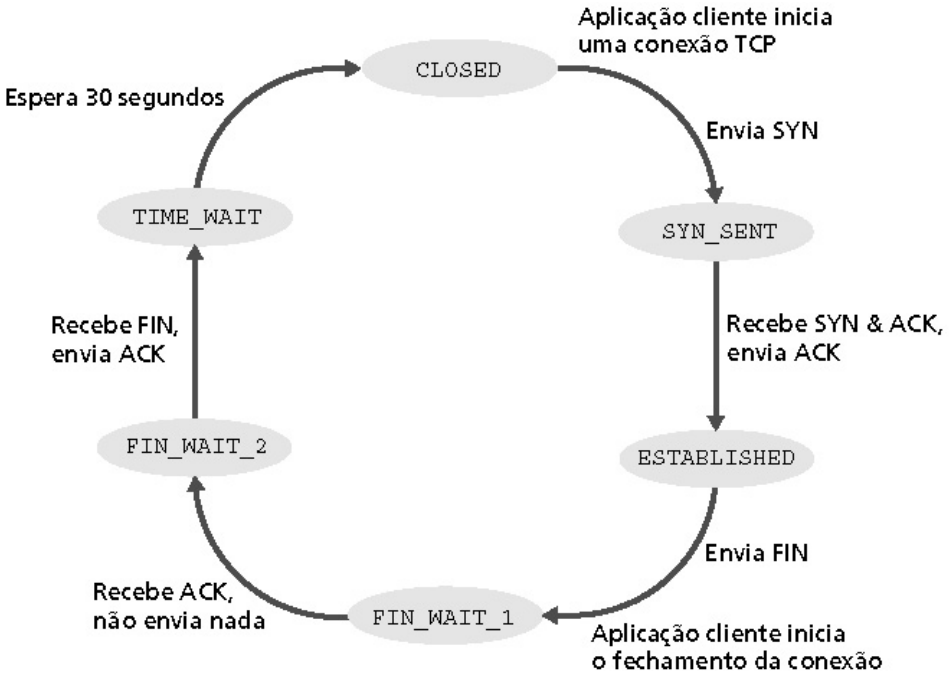
- Entra “espera temporizada” - vai responder com ACK a FINs recebidos

Passo 4: servidor, recebe ACK. Conexão fechada

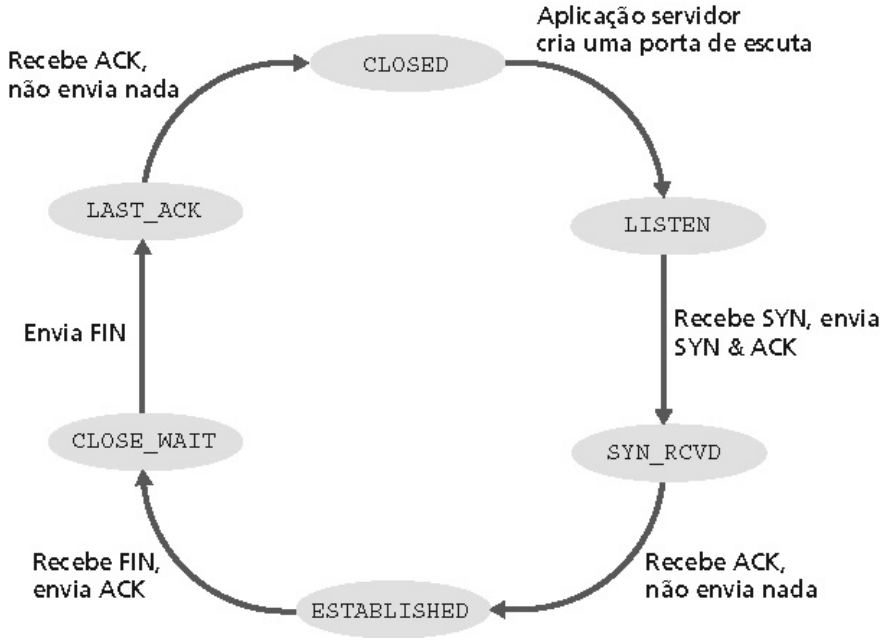
Nota: com uma pequena modificação, pode-se manipular FINs simultâneos



3 Gerenciamento de conexão TCP



Estados do cliente



Estados do servidor