Capítulo 3: Camada de transporte

Objetivos do capítulo:

- entender princípios por trás dos serviços da camada de transporte:
  - multiplexação/demul-tiplexação
  - transferência de dados confiável
  - controle de fluxo
  - controle de congestionamento
- aprender sobre os protocolos da camada de transporte na Internet:
  - UDP: transporte sem conexão
  - TCP: transporte orientado a conexão
  - controle de congestionamento TCP

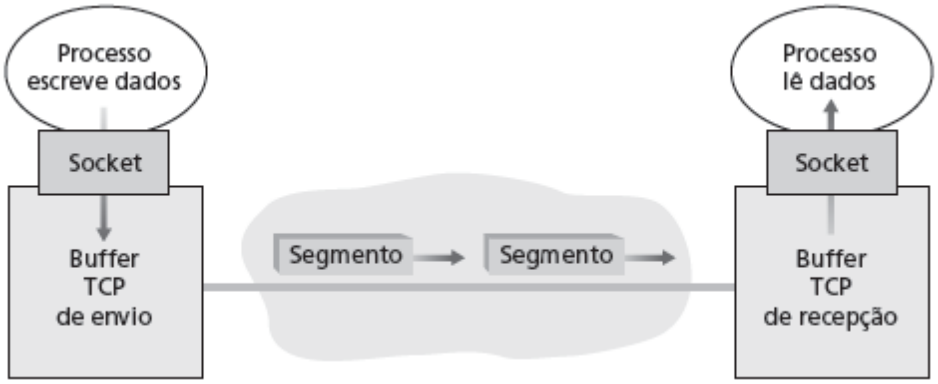
Capítulo 3: Esboço

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

TCP: Visão geral

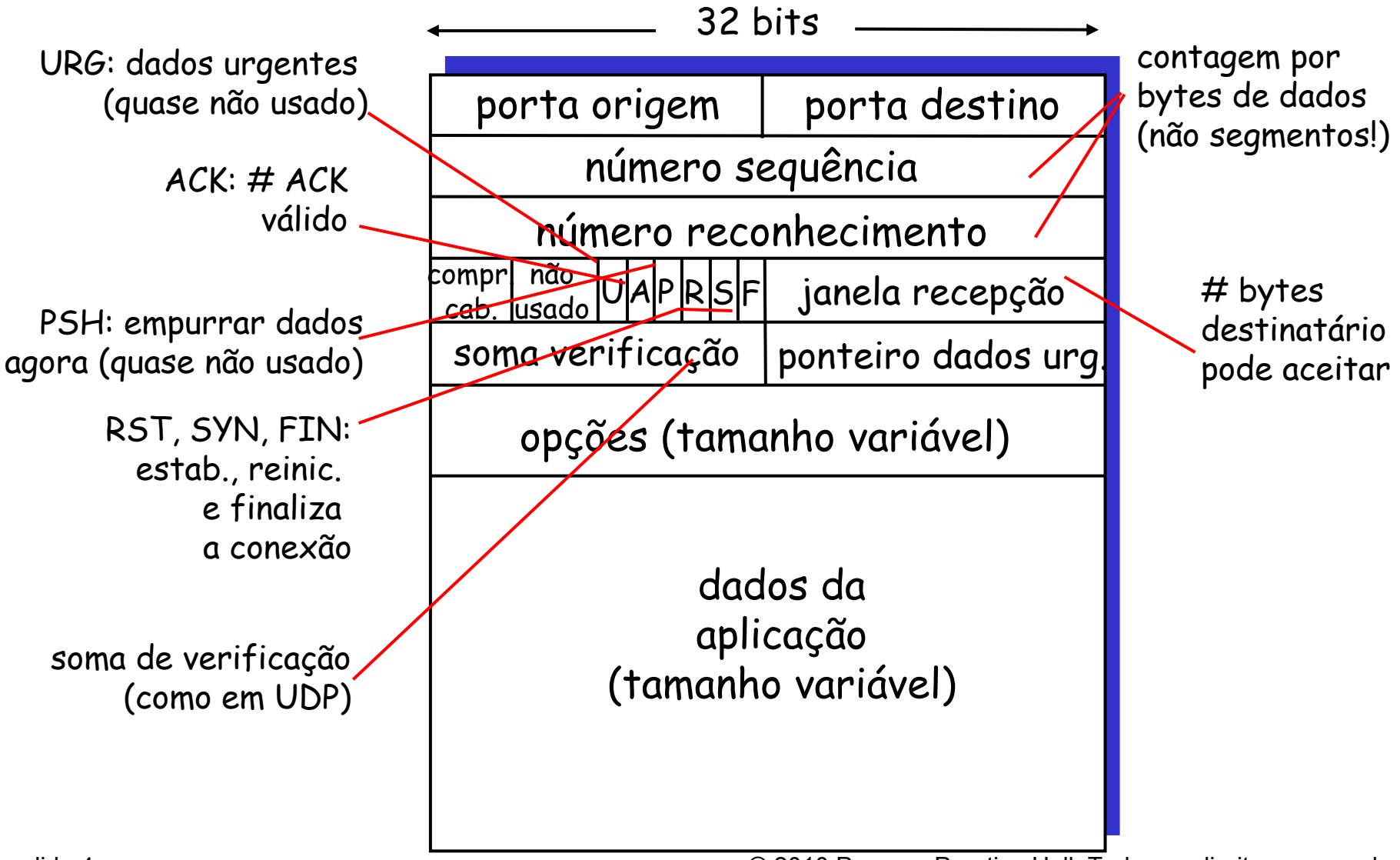
RFCs: 793, 1122, 1323, 2018, 2581

- ponto a ponto:
  - um remetente, um destinatário
- *cadeia de bytes* confiável, em ordem:
  - sem "limites de mensagem"
- paralelismo:
  - congestionamento TCP e controle de fluxo definem tamanho da janela
- *buffers de envio & recepção*



- dados full duplex:
  - dados bidirecionais fluem na mesma conexão
  - MSS: tamanho máximo do segmento
- orientado a conexão:
  - apresentação (troca de msgs de controle) inicia estado do remetente e destinatário antes da troca de dados
- fluxo controlado:
  - remetente não sobrecarrega destinatário

Estrutura do segmento TCP



#s sequência e ACKs do TCP

#'s de sequência:

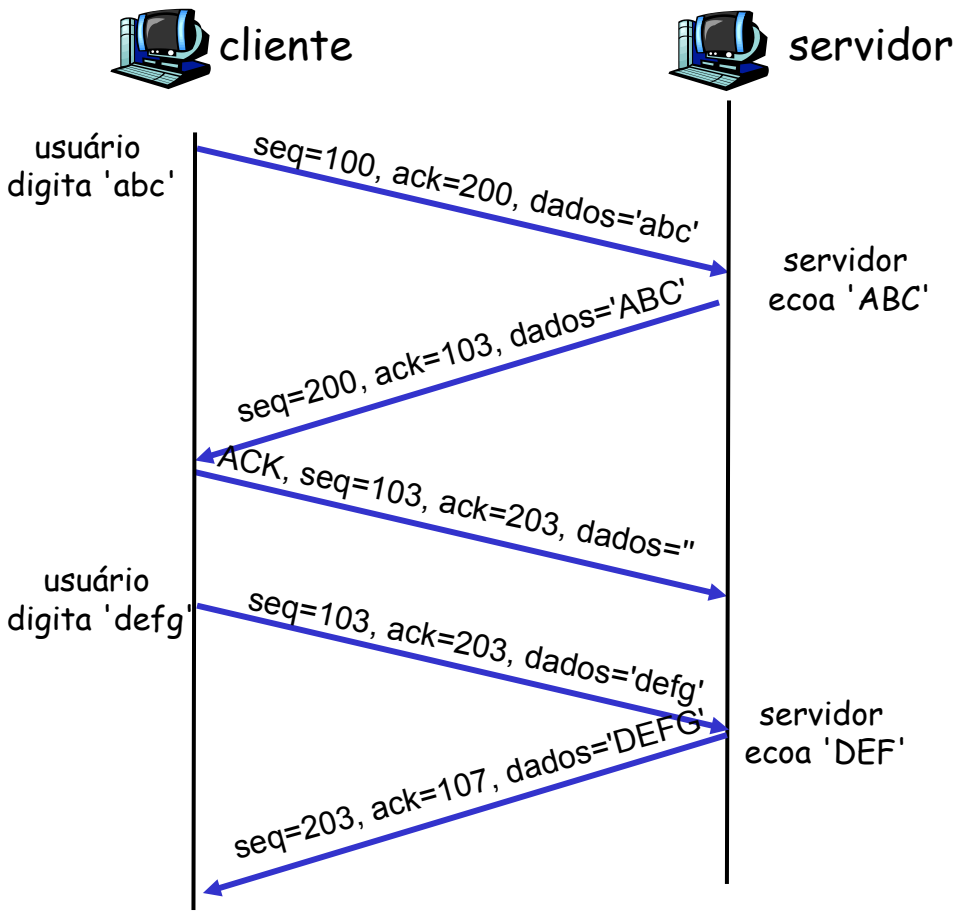
- "número" na cadeia de bytes do 1º byte nos dados do segmento

ACKs:

- # seq do próximo byte esperado do outro lado
- ACK cumulativo

P: como o destinatário trata segmentos fora de ordem

- R: TCP não diz – a critério do implementador



cenário telnet simples

Tempo de ida e volta e *timeout* do TCP

P: Como definir o valor de *timeout* do TCP?

- maior que RTT
  - mas RTT varia
- muito curto: *timeout* prematuro
  - retransmissões desnecessárias
- muito longo: baixa reação a perda de segmento

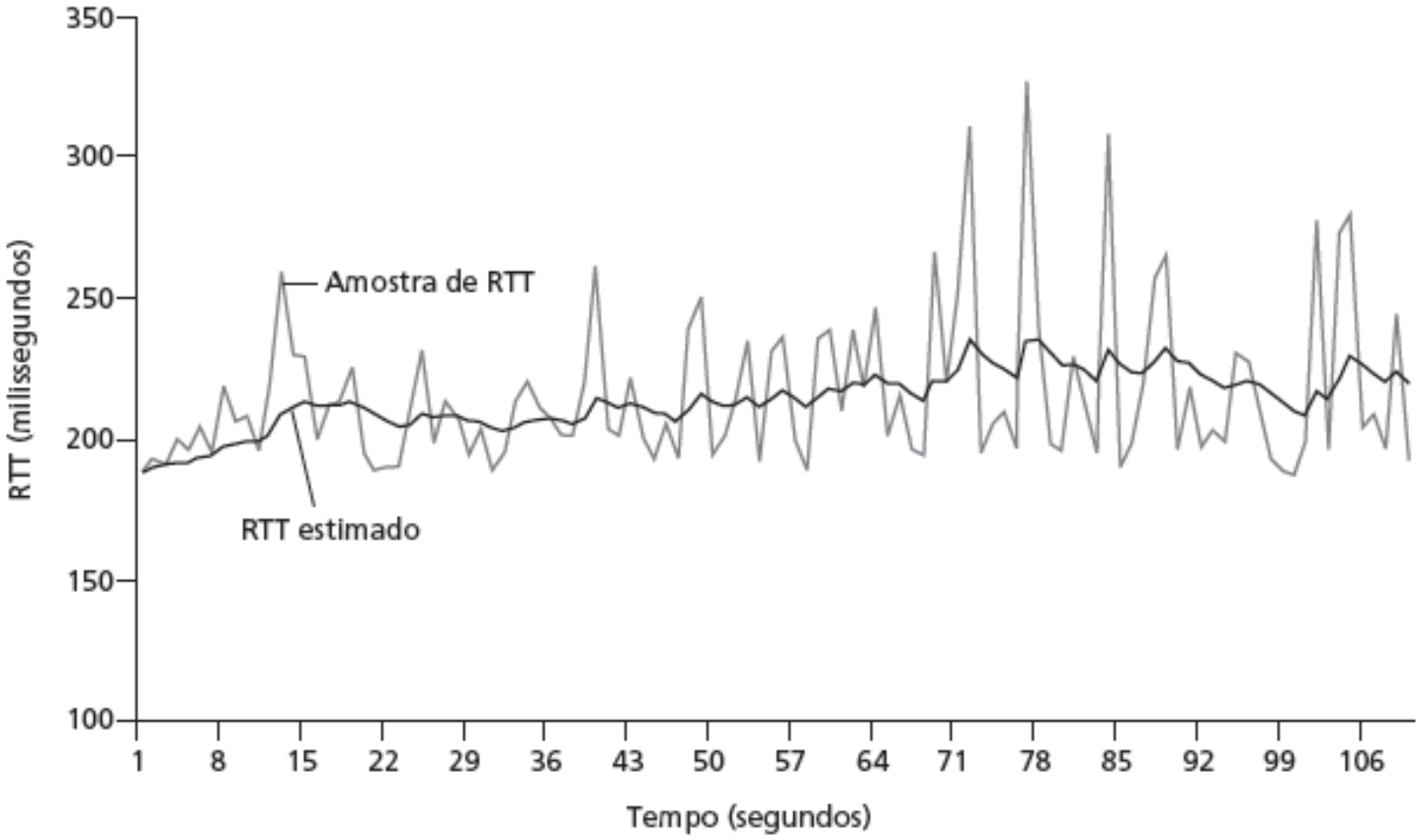
P: Como estimar o RTT?

- **SampleRTT**: tempo medido da transmissão do segmento até receber o ACK
  - ignora retransmissões
- **SampleRTT** variará; queremos RTT estimado "mais estável"
  - média de várias medições recentes, não apenas **SampleRTT** atual

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

- média móvel exponencial ponderada
- influência da amostra passada diminui exponencialmente rápido
- valor típico: **θ** = 0,125

Amostras de RTTs estimados:



Tempo de ida e volta e *timeout* do TCP

definindo o *timeout*

- **`EstimatedRTT`** mais "margem de segurança"
  - grande variação em **`EstimatedRTT`** -> maior margem de seg.
- primeira estimativa do quanto SampleRTT se desvia de EstimatedRTT:

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

(geralmente, $\beta = 0,25$)

depois definir intervalo de *timeout*

$$TimeoutInterval = EstimatedRTT + 4*DevRTT$$

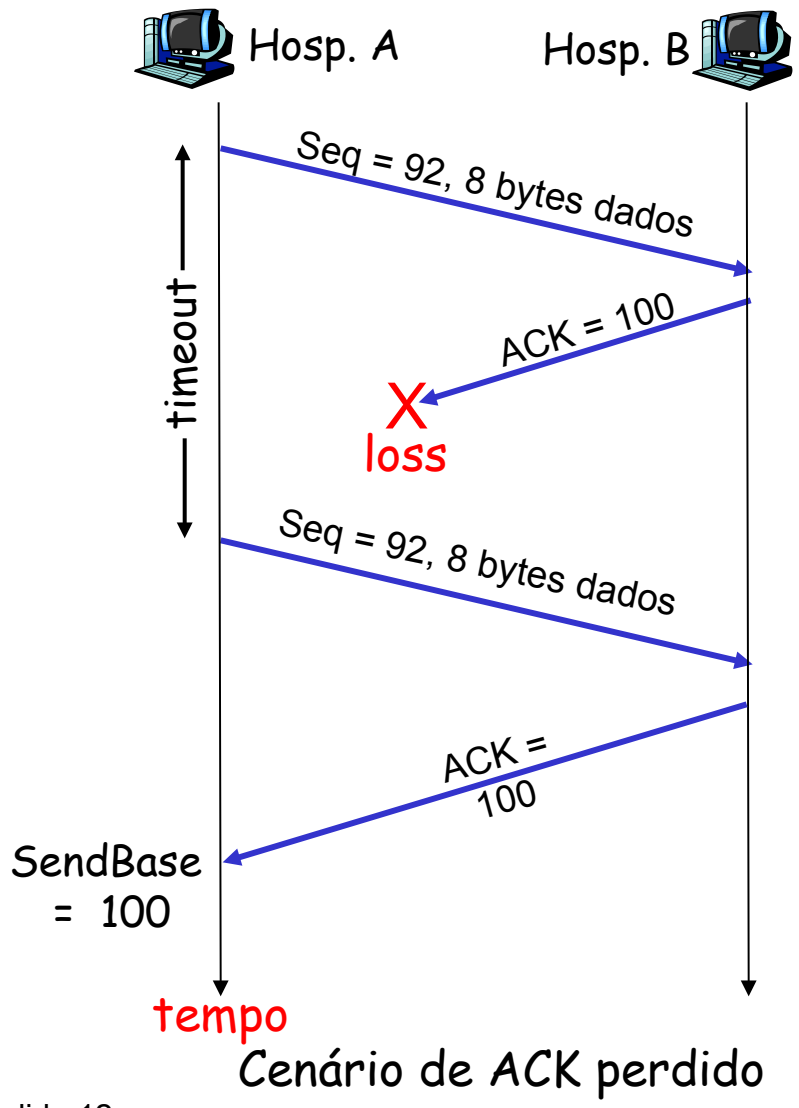
Capítulo 3: Esboço

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

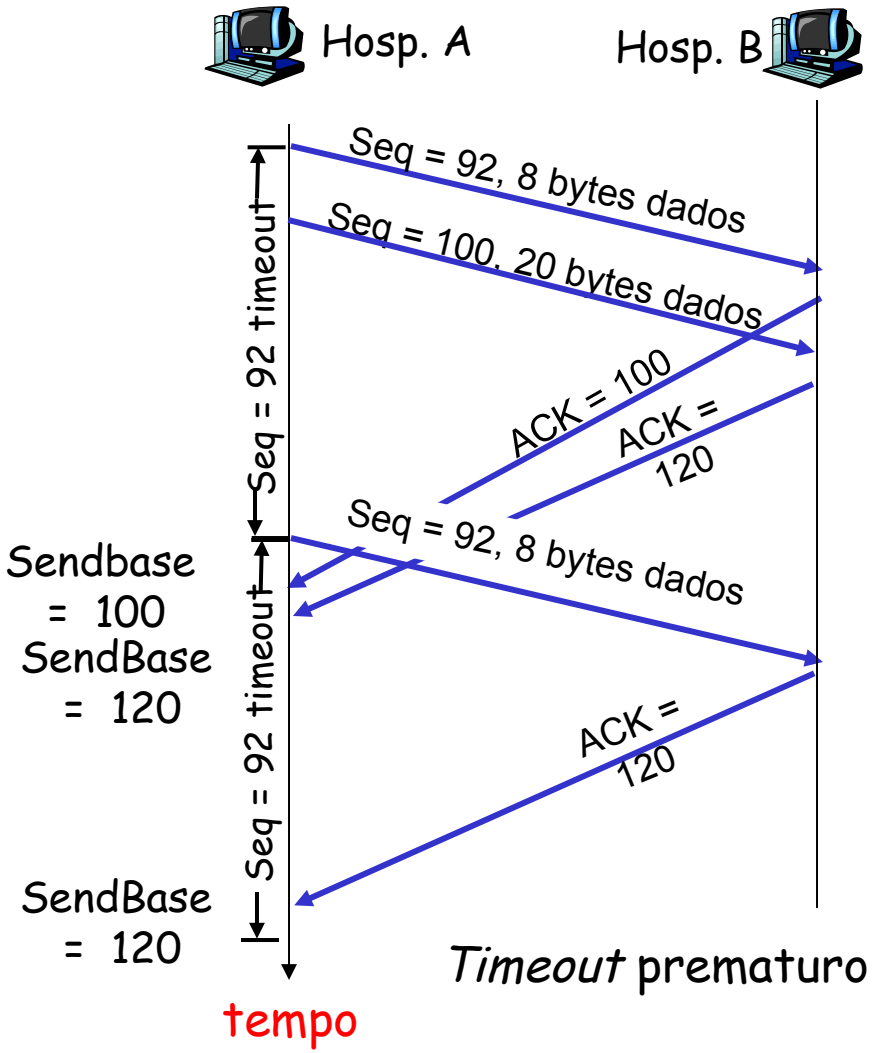
Transferência confiável de dados no TCP

- TCP cria serviço rdt em cima do serviço não confiável do IP
- segmentos em paralelo
- ACKs cumulativos
- TCP usa único temporizador de retransmissão
- retransmissões são disparadas por:
  - eventos de *timeout*
  - ACKs duplicados
- inicialmente, vamos considerar remetente TCP simplificado:
  - ignora controle de fluxo, controle de congestionamento

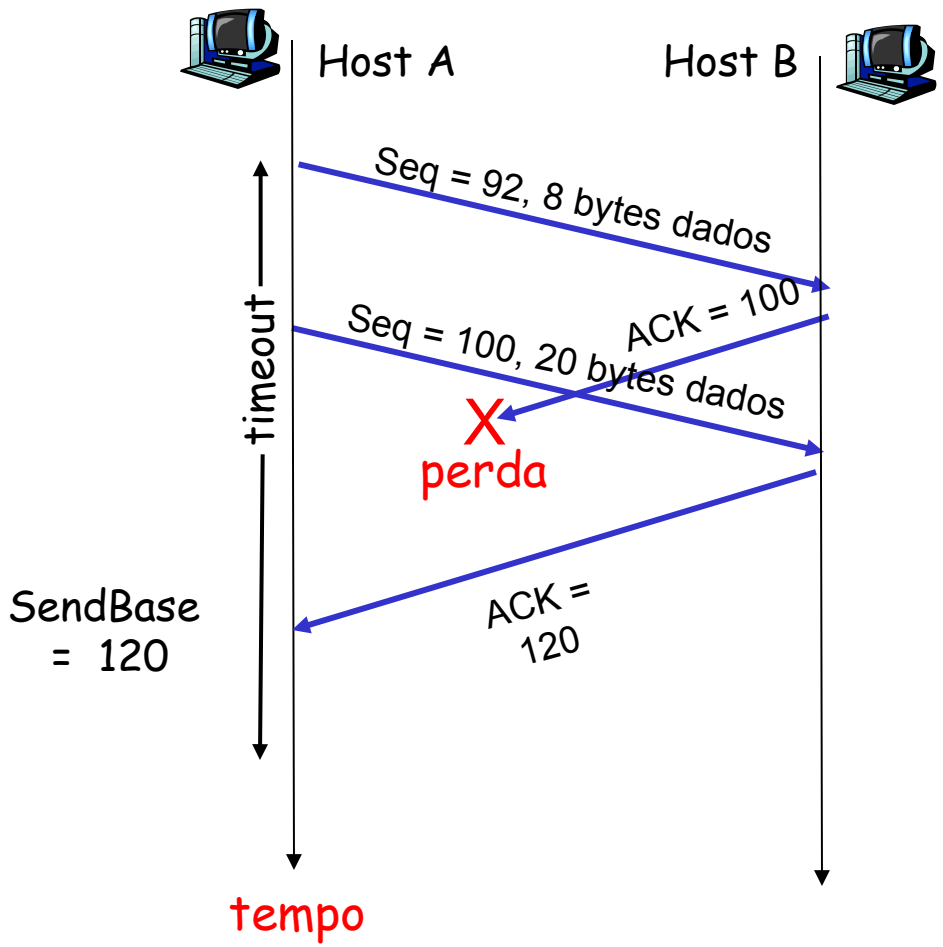
TCP: cenários de retransmissão



Cenário de ACK perdido



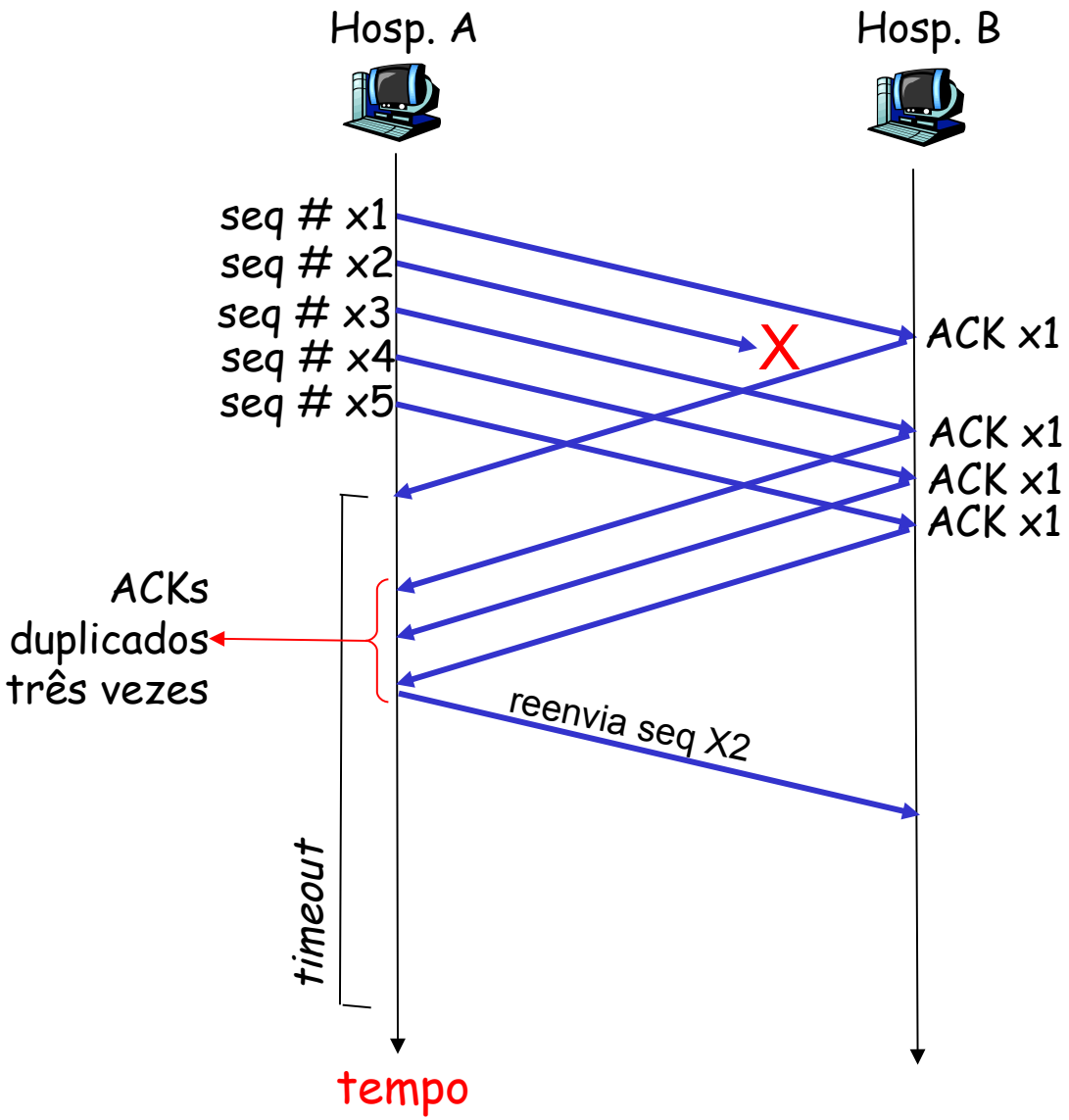
Timeout prematuro



Cenário ACK cumulativo

Retransmissão rápida

- período de *timeout* relativamente grande:
  - longo atraso antes de reenviar pacote perdido
- detecta segmentos perdidos por meio de ACKs duplicados
  - remetente geralmente envia muitos segmentos um após o outro
  - se segmento for perdido, provavelmente haverá muitos ACKs duplicados para esse segmento
- se remetente recebe 3 ACKs para os mesmos dados, ele supõe que segmento após dados com ACK foi perdido:
  - retransmissão rápida: reenvia segmento antes que o temporizador expire

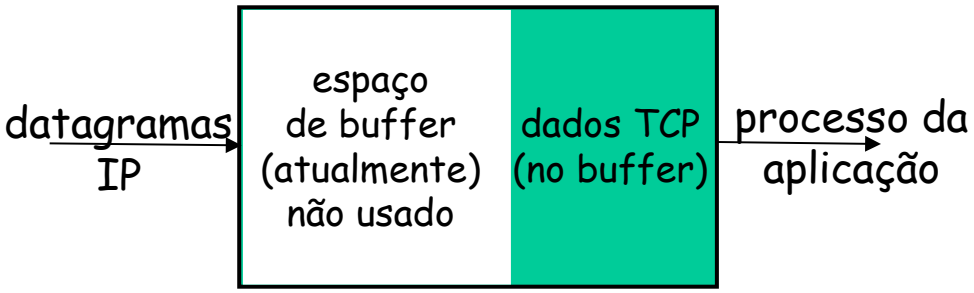


Capítulo 3: Esboço

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

Controle de fluxo TCP

- lado receptor da conexão TCP tem um buffer de recepção:



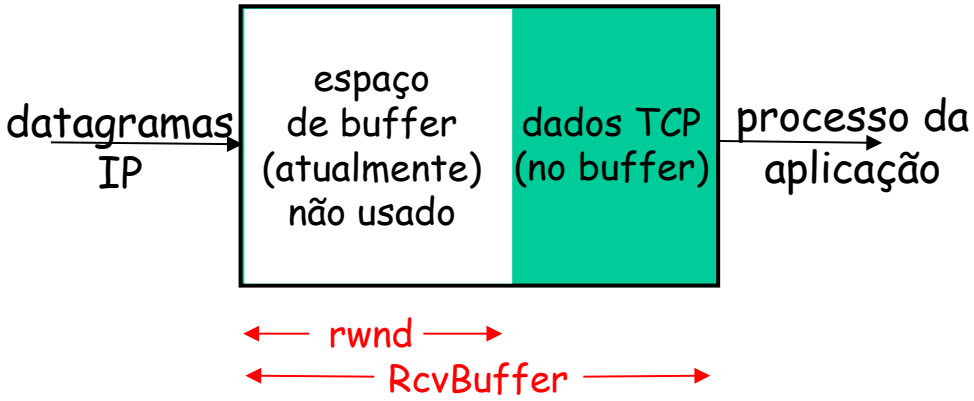
- processo da aplicação pode ser lento na leitura do buffer

controle de fluxo

remetente não estourará buffer do destinatário transmitindo muitos dados muito rapidamente

- *serviço de compatibilização de velocidades:* compatibiliza a taxa de envio do remetente com a de leitura da aplicação receptora

Controle de fluxo TCP: como funciona



(suponha que destinatário TCP descarte segmentos fora de ordem)

- espaço de buffer não usado:

  = **rwnd**

  = **RcvBuffer-[LastByteRcvd - LastByteRead]**

- destinatário: anuncia espaço de buffer não usado incluindo valor de `rwnd` no cabeçalho do segmento
- remetente: limita # de bytes sem ACK a `rwnd`
  - garante que buffer do destinatário não estoura

Capítulo 3: Esboço

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

Gerenciamento da conexão TCP

lembre-se: Remetente e destinatário TCP estabelecem "conexão" antes que troquem segmentos dados

- inicializa variáveis TCP:
  - #s seq.:
  - buffers, informação de controle de fluxo (p. e. **RcvWindow**)
- *cliente:* inicia a conexão

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

- *servidor:* contactado pelo cliente

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

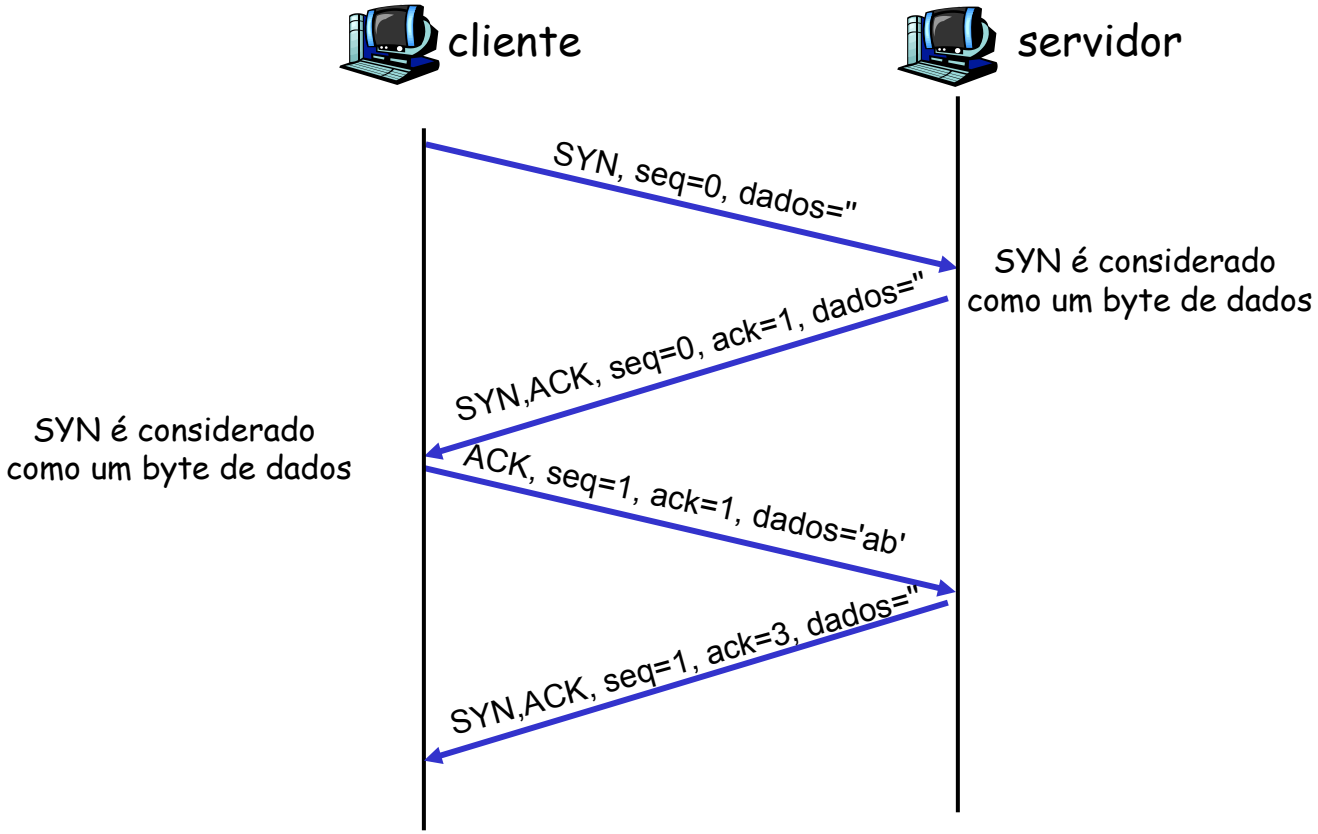
apresentação de 3 vias:

etapa 1: cliente envia segmento SYN do TCP ao servidor
- especifica # seq. inicial
- sem dados

etapa 2: servidor recebe SYN, responde com segmento SYNACK
- servidor aloca buffers
- especifica # seq. inicial do servidor

etapa 3: cliente recebe SYNACK, responde com segmento ACK, que pode conter dados

Gerenciamento da conexão TCP



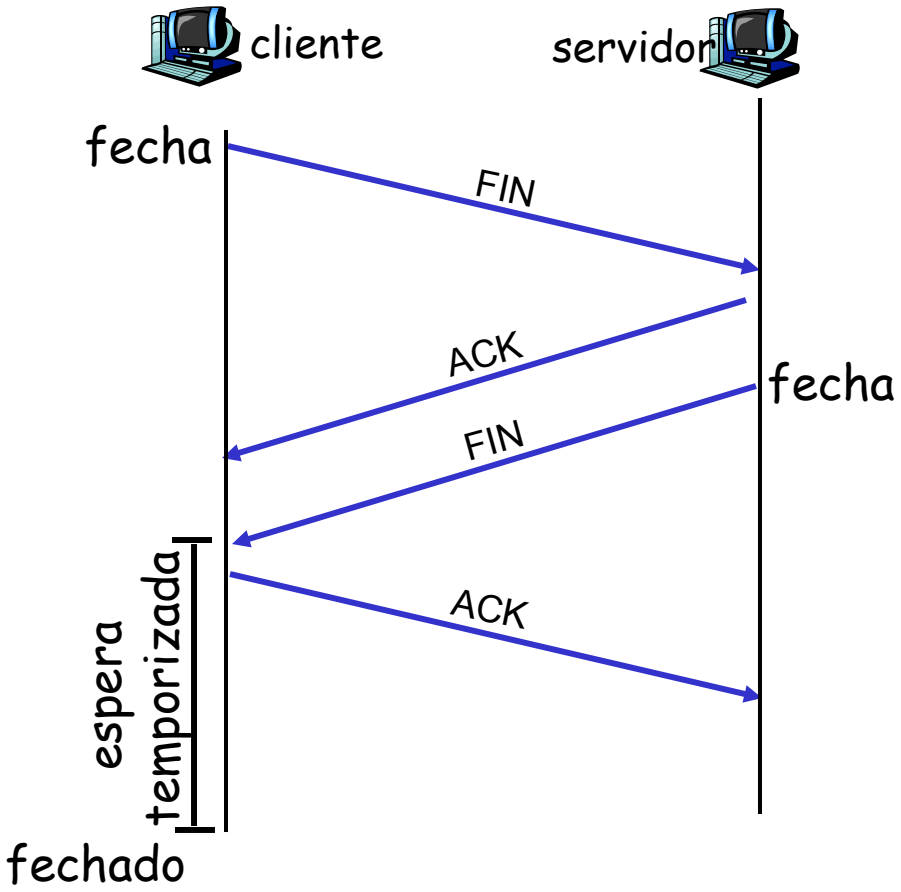
fechando uma conexão:

cliente fecha socket:

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

etapa 1: sistema final do cliente envia segmento de controle TCP FIN ao servidor

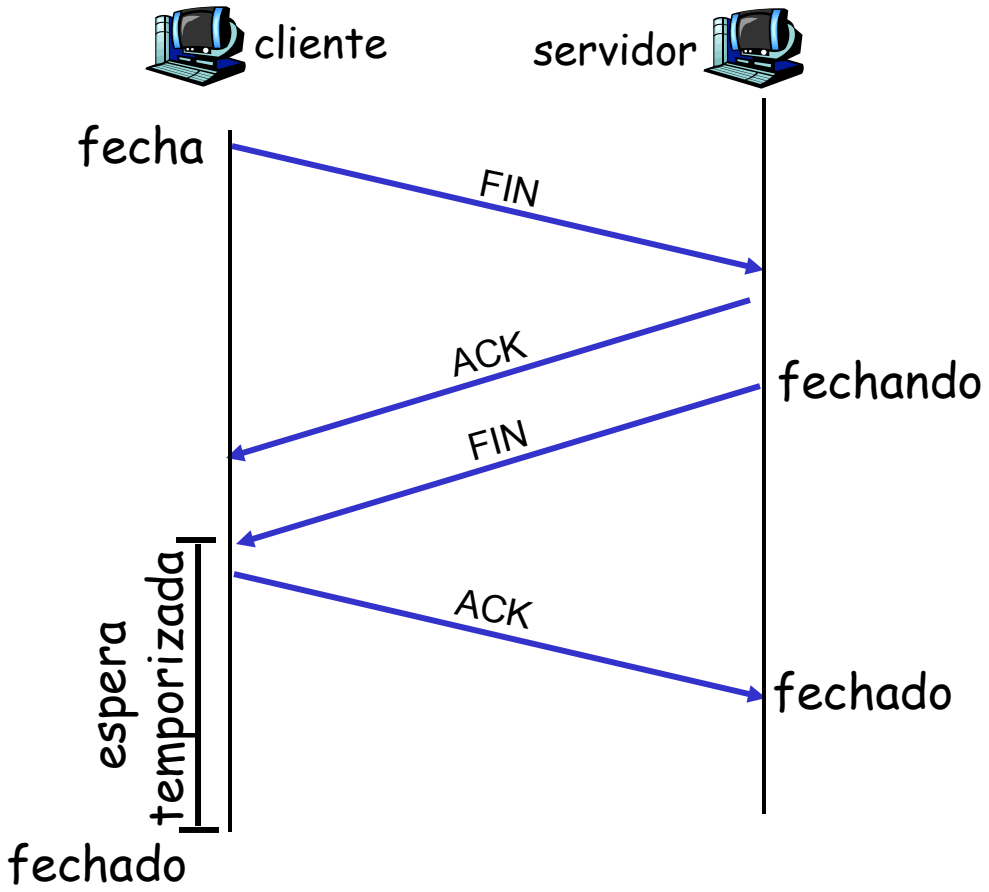
etapa 2: servidor recebe FIN, responde com ACK. Fecha conexão, envia FIN.

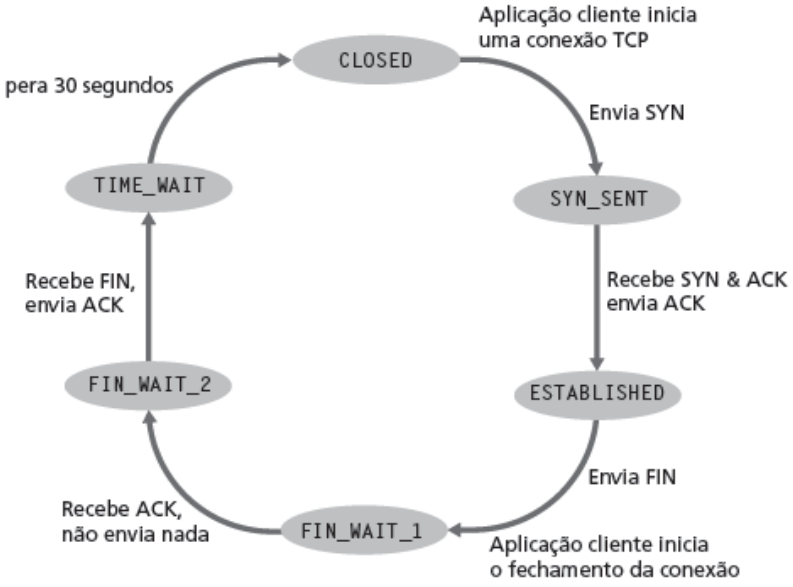


etapa 3: cliente recebe FIN, responde com ACK

- entra em "espera temporizada" – responderá com ACK aos FINs recebidos

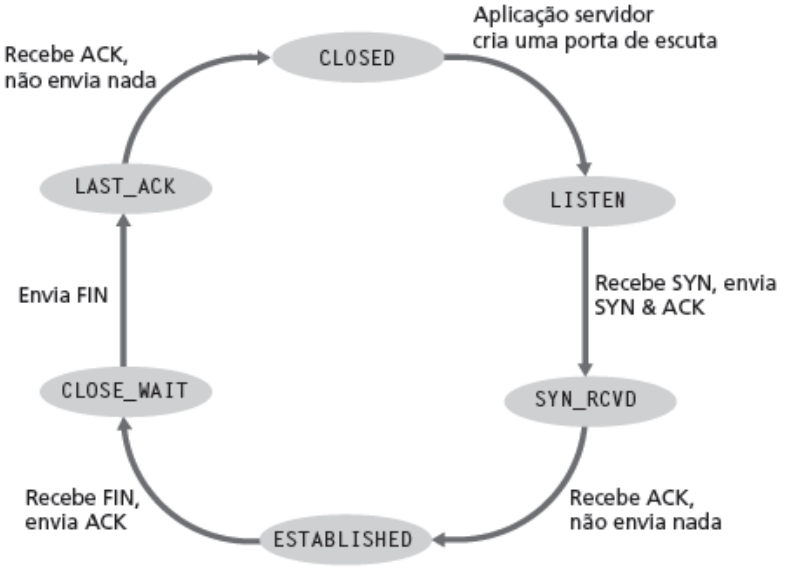
etapa 4: servidor recebe ACK - conexão fechada





ciclo de vida do cliente TCP

ciclo de vida do servidor TCP



Capítulo 3: Esboço

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

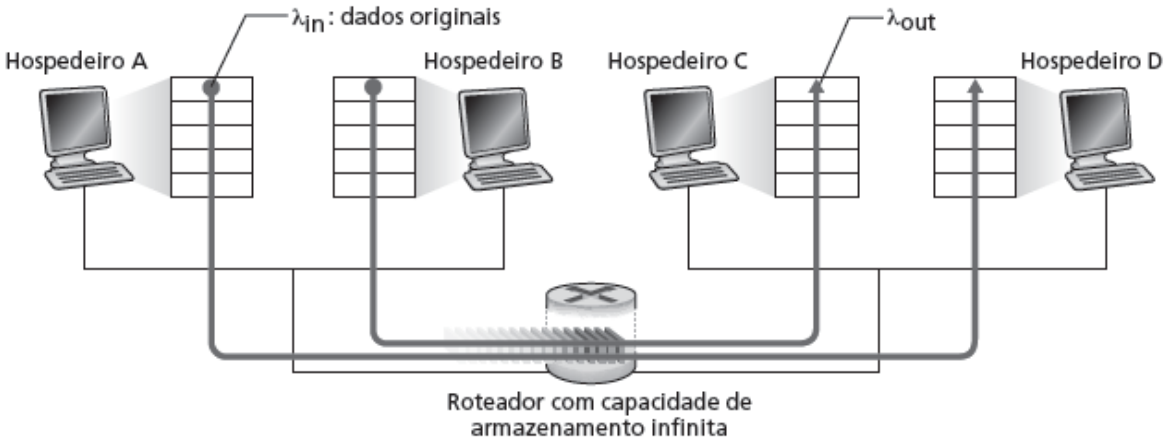
Princípios de controle de congestionamento

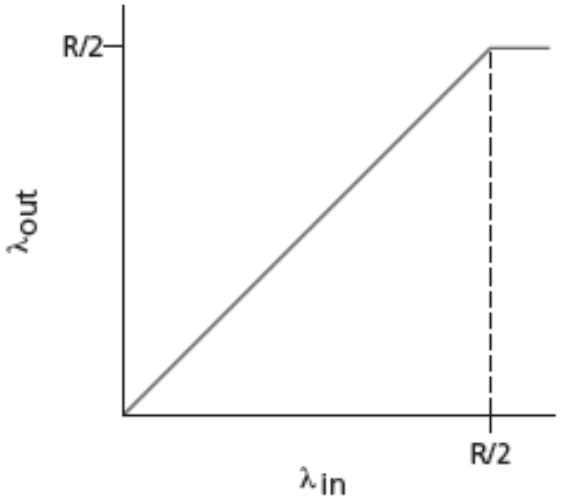
Caso Real (Jacobson and Karels, 1988):

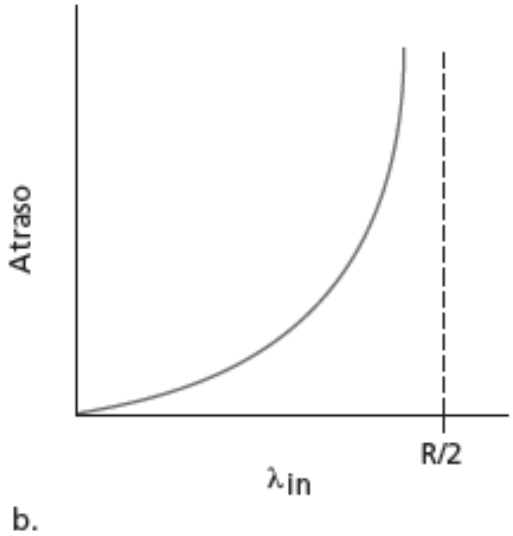
- 1986: colapso por congestionamento na internet
- Informalmente: "muitas fontes enviando muitos dados muito rápido para a *rede* tratar"
- Queda no throughput de dados: de 32Kbps para 40bps
- Por que ocorreu tal colapso?
  - Retransmissões
- Pode-se programar a rede para funcionar bem em tal situação de colapso?
  - Controle de Congestionamento

Causas/custos do congestionamento: cenário 1

- dois remetentes, dois destinatários
- um roteador, buffers infinitos
- sem retransmissão



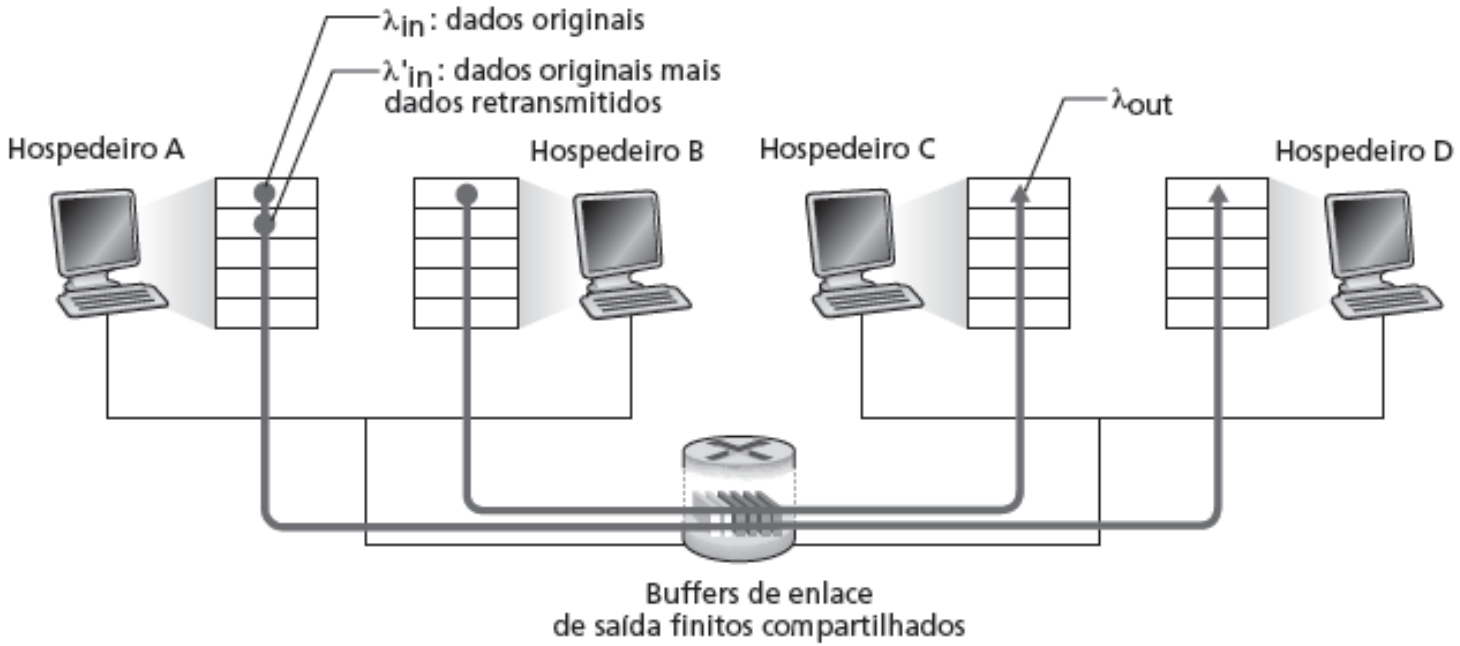




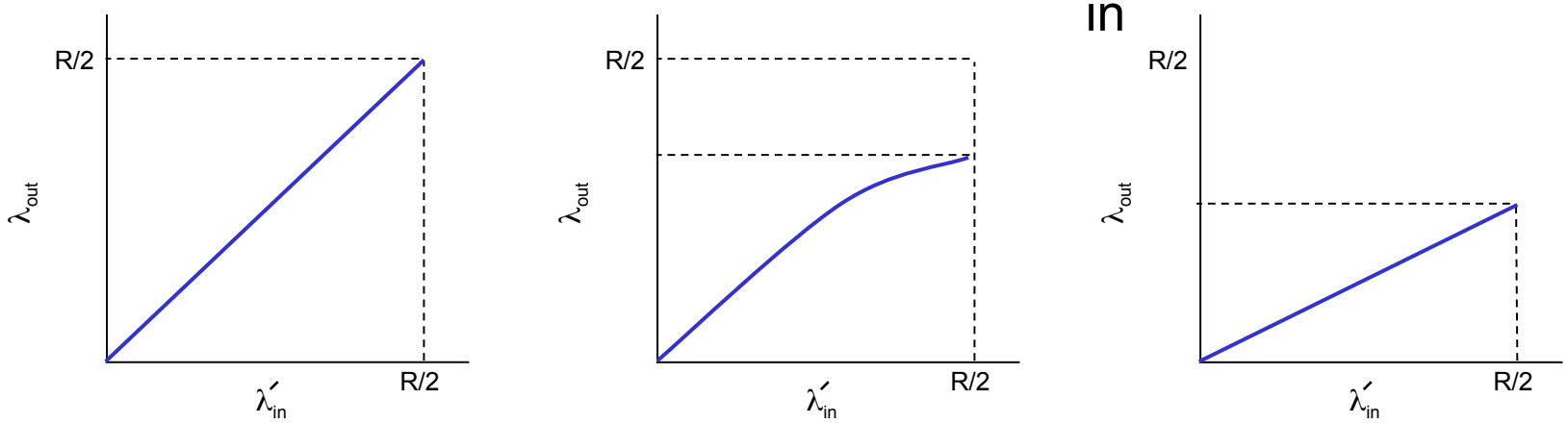
- grandes atrasos quando congestionado
- vazão máxima alcançável

Causas/custos do congestionamento: cenário 2

- um roteador, buffers *finitos*
- retransmissão do pacote perdido pelo remetente



- a. sem retransmissão: $\lambda'_{in} = \lambda_{out}$ (vazão)
- b. retransmissão "perfeita" apenas quando há perda: $\lambda'_{in} > \lambda_{out}$
- c. retransmissão do pacote adiado (não perdido) torna λ'_{in} maior (que o caso perfeito) para o mesmo λ_{in}



a. b. c.

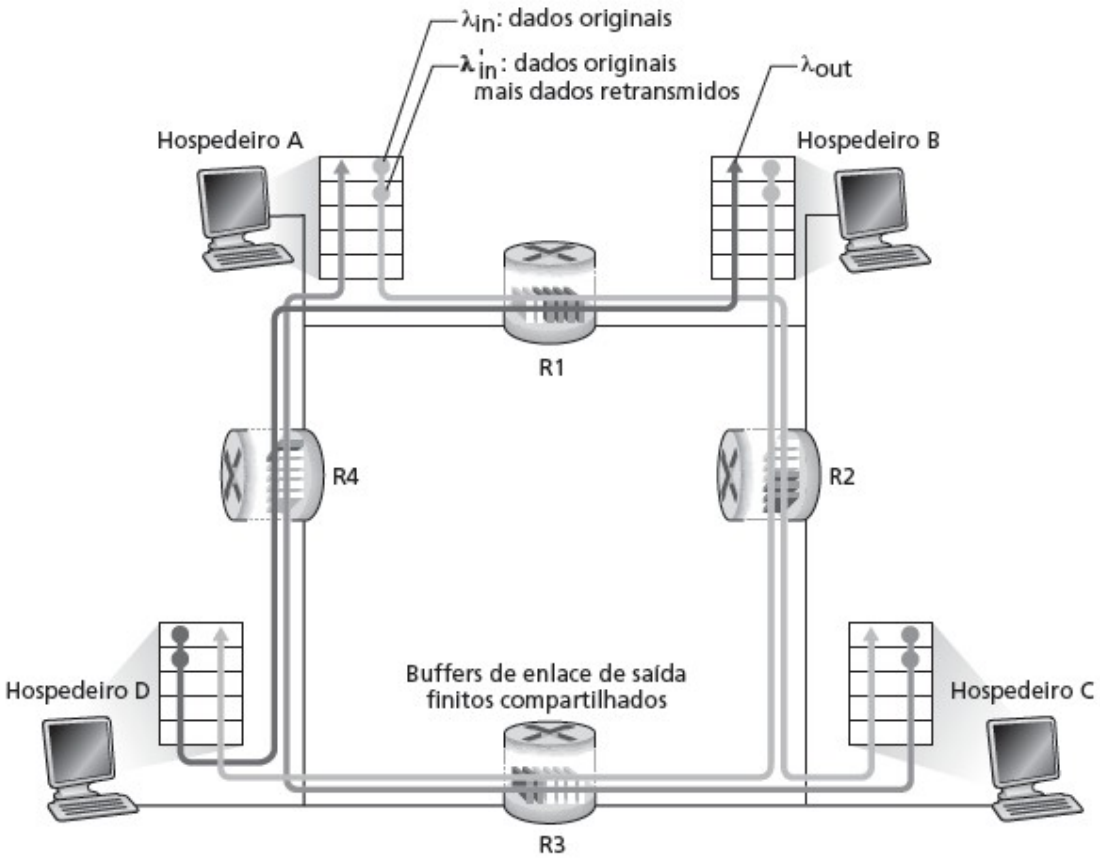
"custos" do congestionamento:

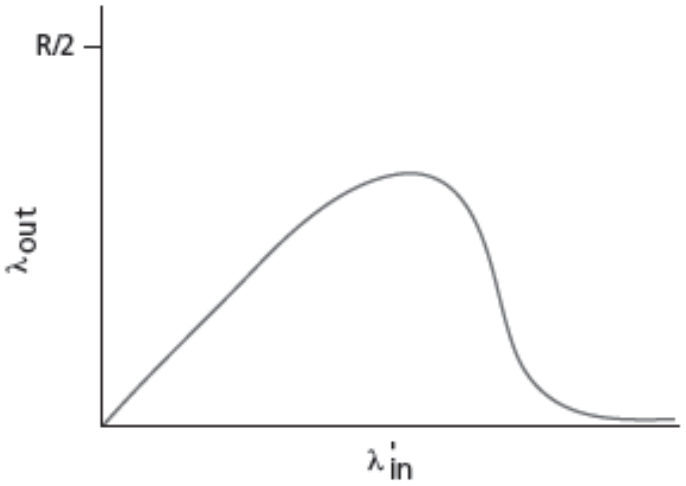
- mais trabalho (retransmissão) para determinada "vazão"
- retransmissões desnecessárias: enlace transporta várias cópias do pacote

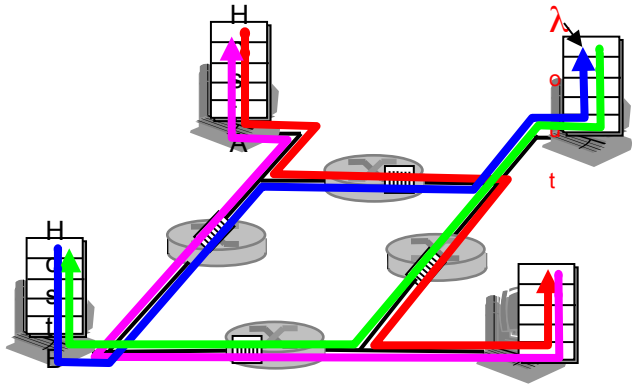
Causas/custos do congestionamento: cenário 3

- quatro remetentes
- caminhos com vários saltos
- *timeout*/retransmissão

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







outro "custo" do congestionamento:

- quando pacote é descartado, qualquer capacidade de transmissão "upstream" usada para esse pacote foi desperdiçada!

Técnicas para controle de congestionamento

duas técnicas amplas para controle de congestionamento:

controle de congestionamento fim a fim:

- nenhum feedback explícito da rede
- congestionamento deduzido da perda e atraso observados do sistema final
- técnica tomada pelo TCP

controle de congestionamento assistido pela rede:

- roteadores oferecem feedback aos sistemas finais
  - único bit indicando congestionamento
  - taxa explícita que o remetente deve enviar no enlace de saída