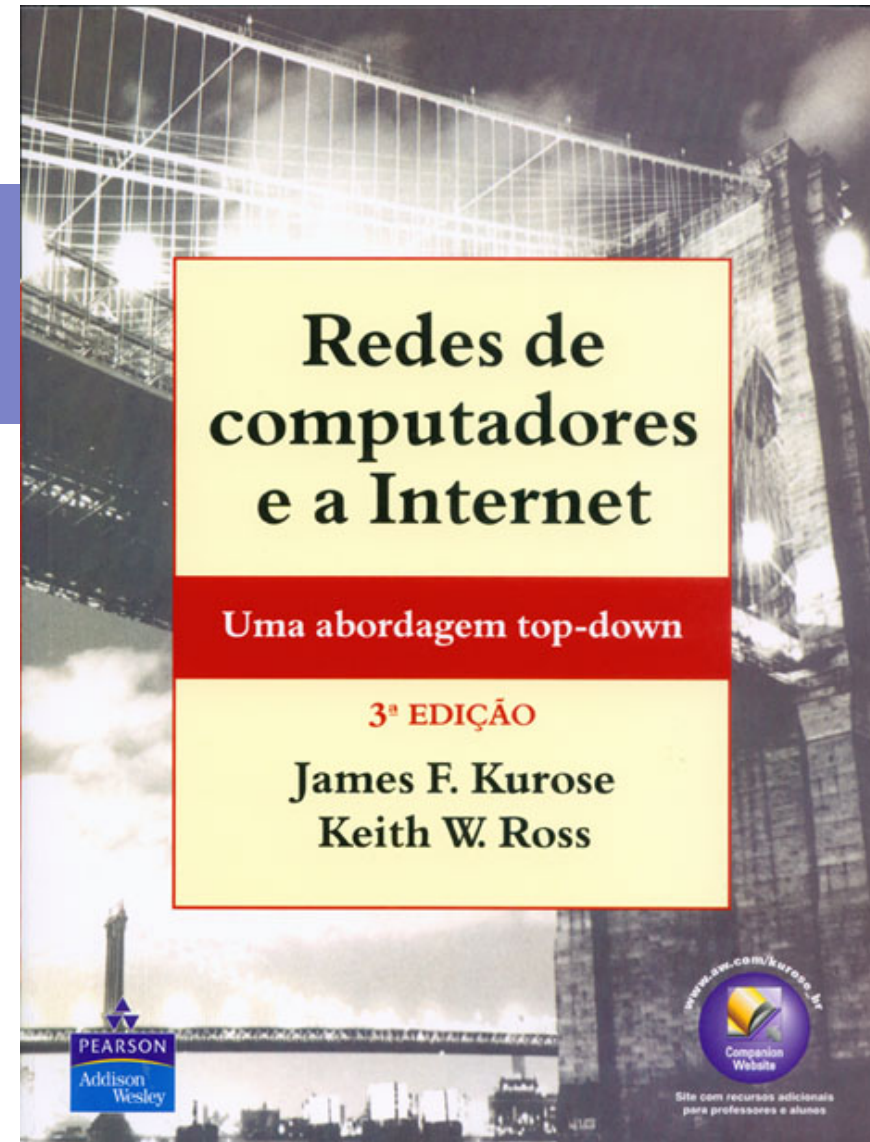


# Redes de computadores e a Internet

## Capítulo 3

### Camada de transporte



# 3 Camada de transporte

## Objetivos do capítulo:

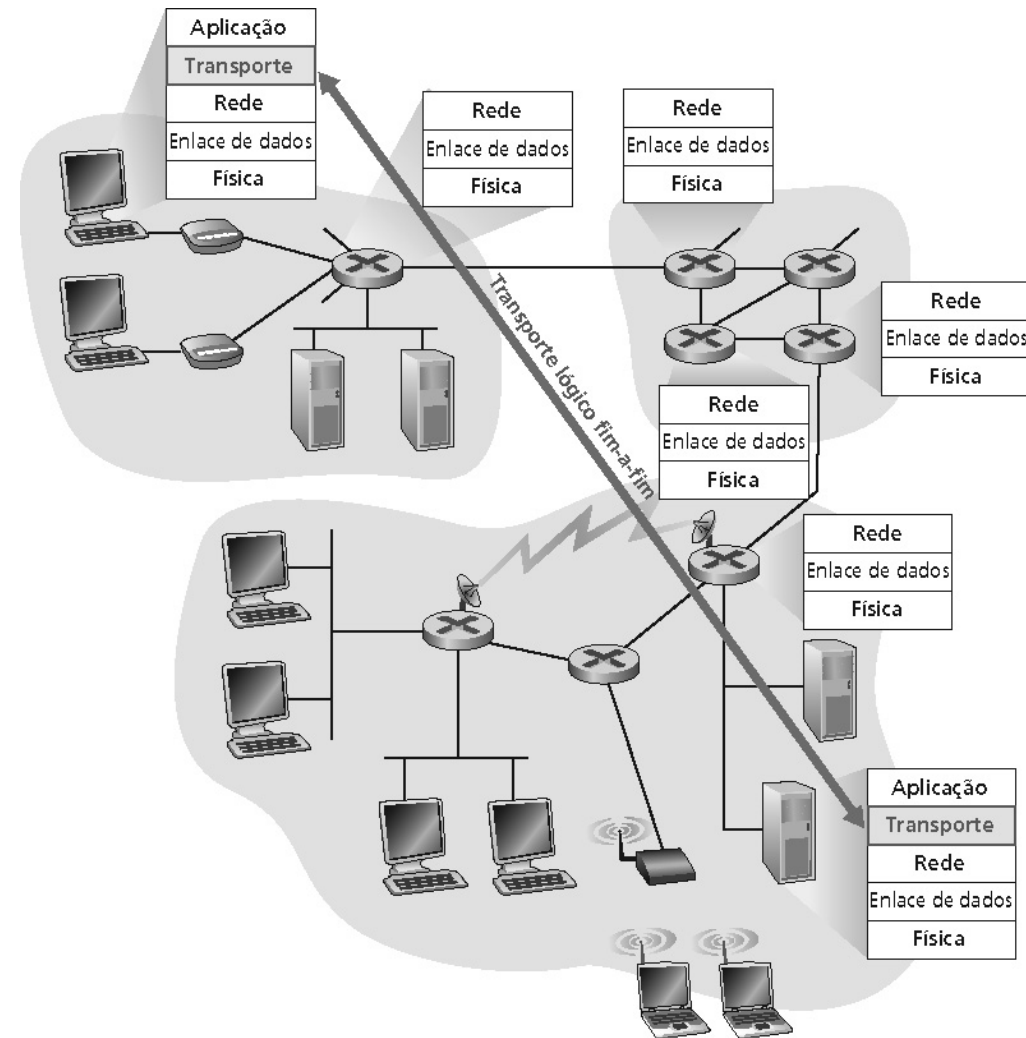
- Entender os princípios por trás dos serviços da camada de transporte:
  - Multiplexação/demultiplexação
  - Transferência de dados confiável
  - Controle de fluxo
  - Controle de congestionamento
- Aprender sobre os protocolos de transporte na Internet:
  - UDP: transporte não orientado à conexão
  - TCP: transporte orientado à conexão
  - Controle de congestionamento do TCP

# 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 Protocolos e serviços de transporte

- Fornecem **comunicação lógica** entre processos de aplicação em diferentes hospedeiros
- Os protocolos de transporte são executados nos sistemas finais
  - Lado emissor: quebra as mensagens da aplicação em segmentos e envia para a camada de rede
  - Lado receptor: remonta os segmentos em mensagens e passa para a camada de aplicação
- Há mais de um protocolo de transporte disponível para as aplicações
  - Internet: TCP e UDP



# 3 Camada de transporte vs. camada de rede

- **Camada de rede:** comunicação lógica entre os hospedeiros
- **Camada de transporte:** comunicação lógica entre os processos
  - Depende dos serviços da camada de rede

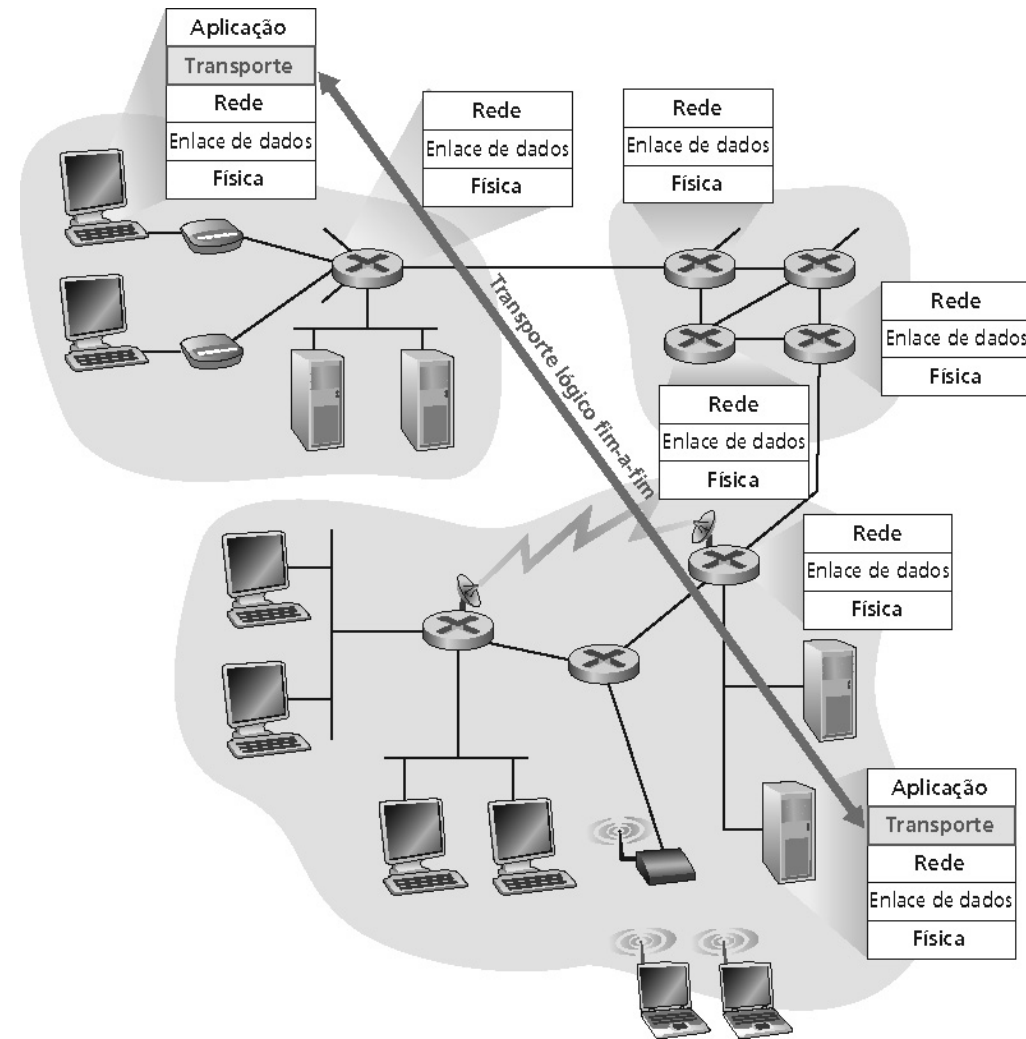
## **Analogia com uma casa familiar:**

12 crianças enviam cartas para 12 crianças

- Processos = crianças
- Mensagens da aplicação = cartas nos envelopes
- Hospedeiros = casas
- Protocolo de transporte = Anna e Bill
- Protocolo da camada de rede = serviço postal

# 3 Protocolos da camada de transporte da Internet

- Confiável, garante ordem de entrega (TCP)
- Controle de congestionamento
  - Controle de fluxo
  - Orientado à conexão
- Não confiável, sem ordem de entrega: UDP
  - Extensão do “melhor esforço” do IP
- Serviços não disponíveis:
  - Garantia a atrasos
  - Garantia de banda



# 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

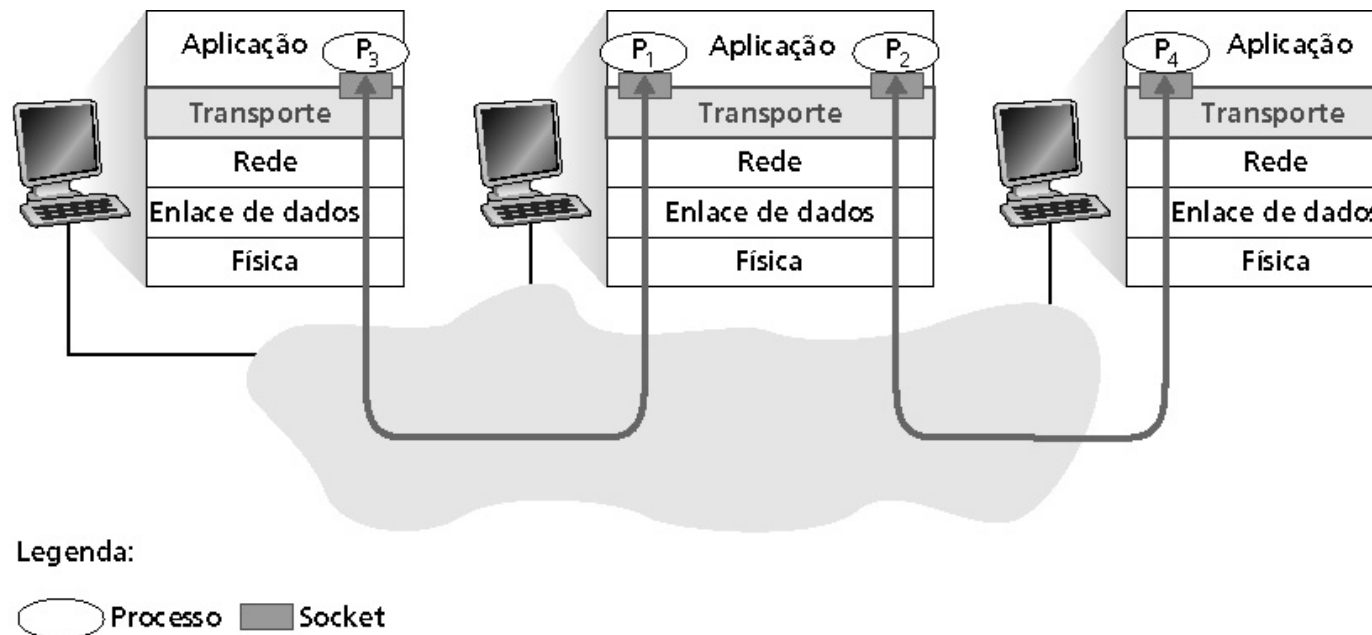
# 3 Multiplexação/demultiplexação

Demultiplexação no hospedeiro receptor:

entrega os segmentos recebidos  
ao socket correto

Multiplexação no hospedeiro emissor:

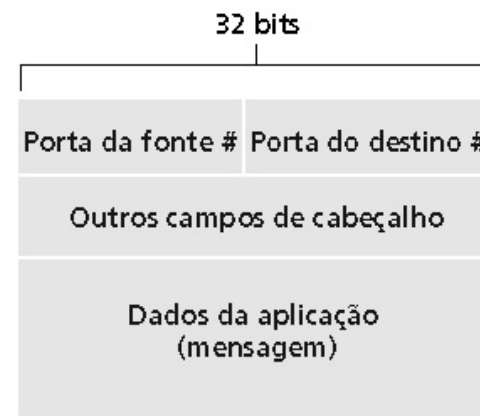
coleta dados de múltiplos sockets,  
envelopa os dados com cabeçalho  
(usado depois para demultiplexação)





# 3 Como funciona a demultiplexação

- **Computador recebe datagramas IP**
  - Cada datagrama possui endereço IP de origem e IP de destino
  - Cada datagrama carrega 1 segmento da camada de transporte
  - Cada segmento possui números de porta de origem e destino (lembre-se: números de porta bem conhecidos para aplicações específicas)
- **O hospedeiro usa endereços IP e números de porta para direcionar o segmento ao socket apropriado**



# 3 Demultiplexação não orientada à conexão

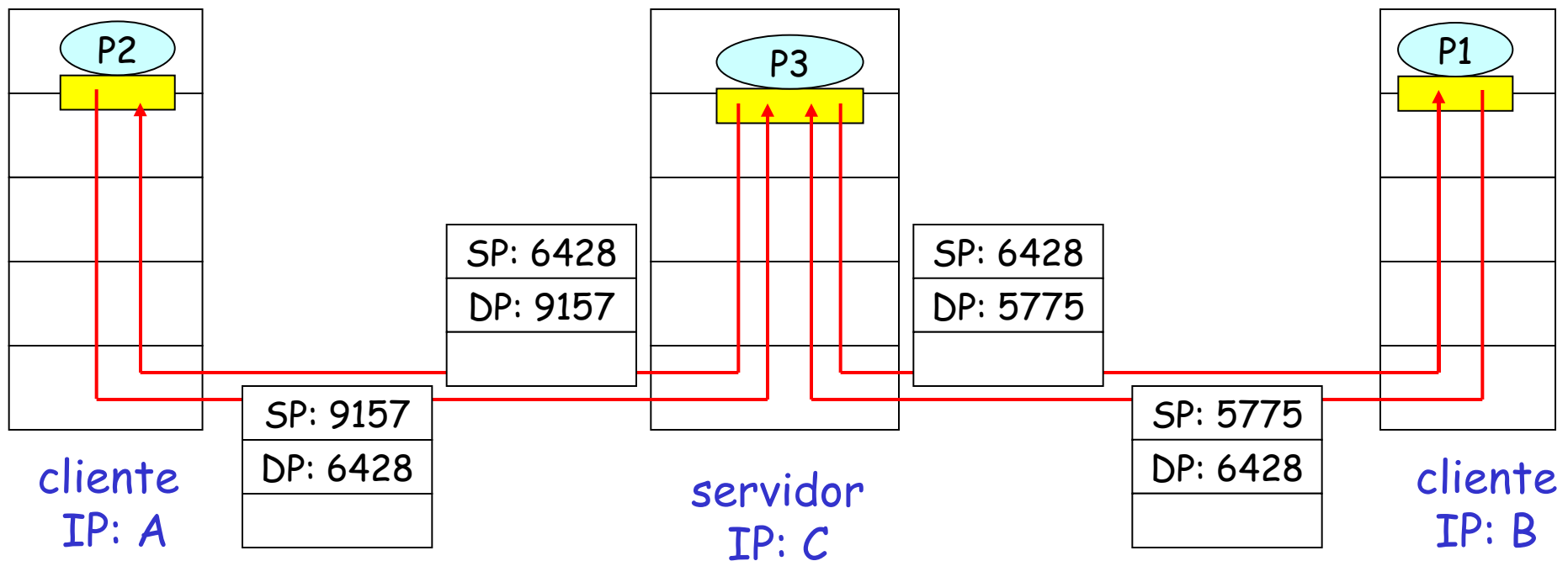
- Cria sockets com números de porta:

```
DatagramSocket mySocket1 = new DatagramSocket(99111);  
DatagramSocket mySocket2 = new DatagramSocket(99222);
```

- Socket UDP identificado por dois valores:  
(endereço IP de destino, número da porta de destino)
- Quando o hospedeiro recebe o segmento UDP:
  - Verifica o número da porta de destino no segmento
  - Direciona o segmento UDP para o socket com este número de porta
- Datagramas com IP de origem diferentes e/ou portas de origem diferentes são direcionados para o mesmo socket

# 3 Demultiplexação não orientada à conexão

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

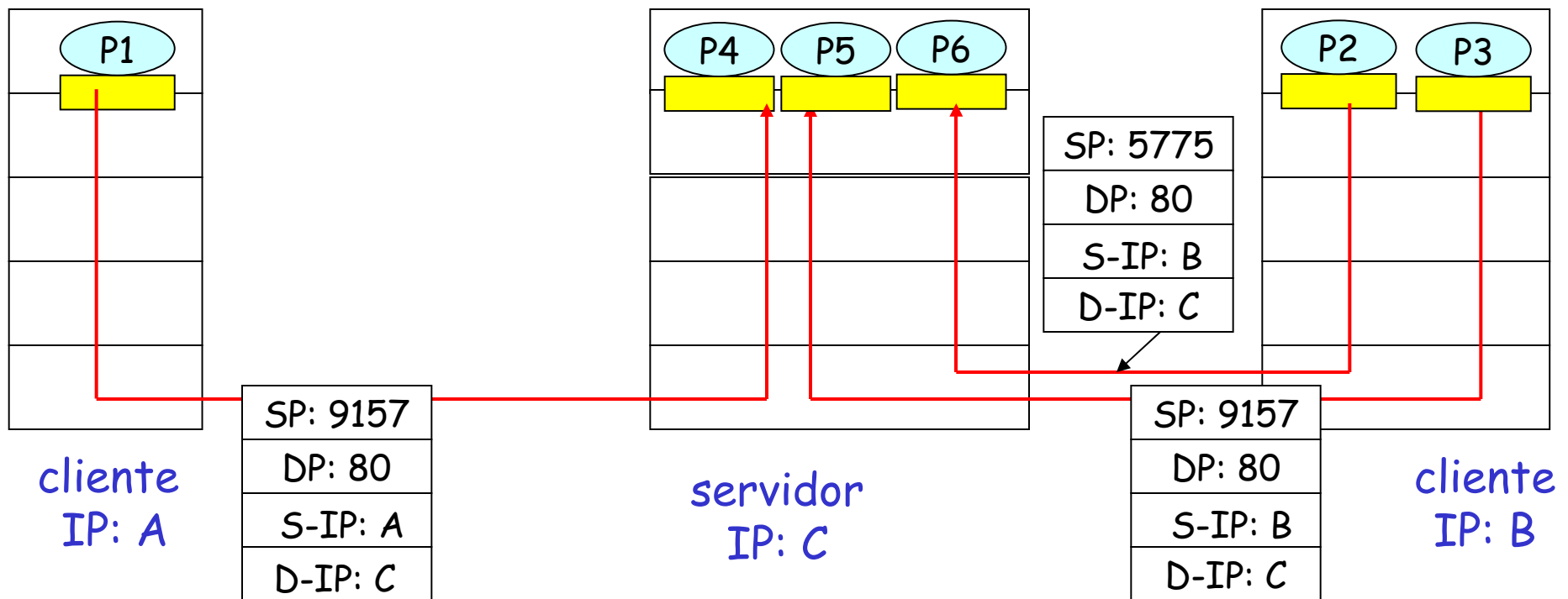


SP fornece o “endereço retorno” 0

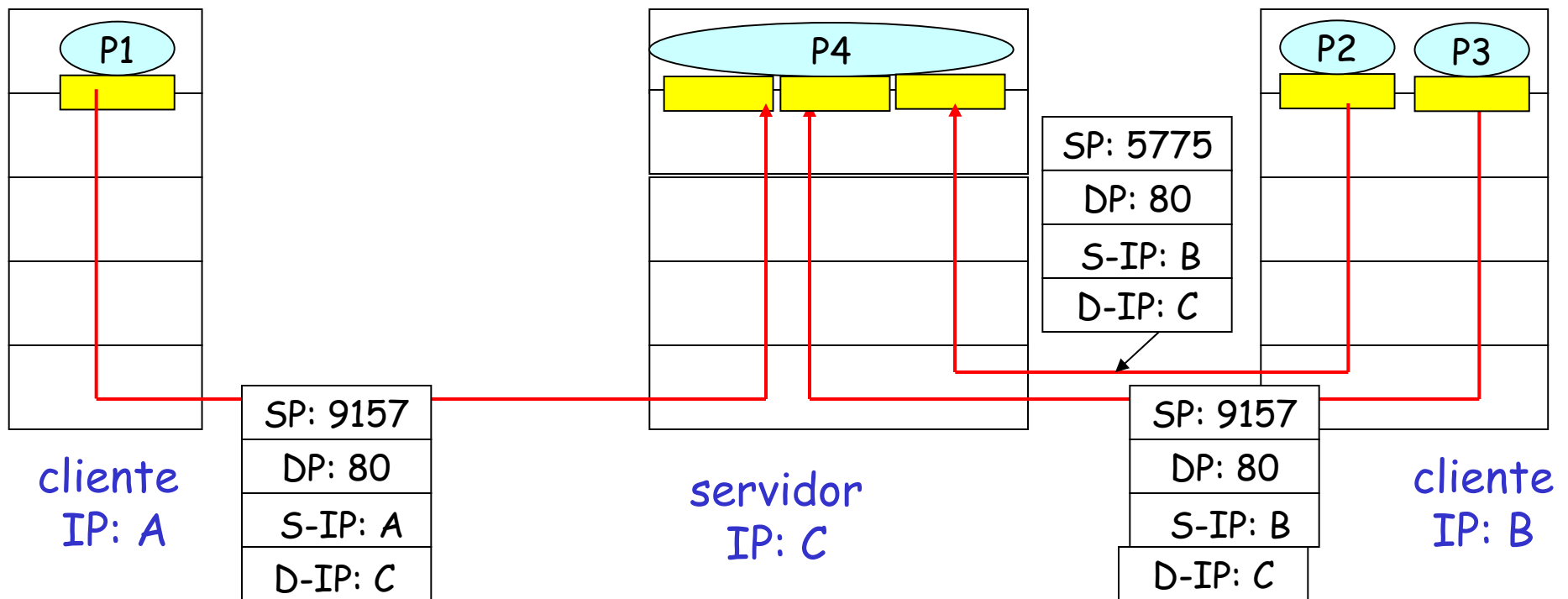
# 3 Demux orientada à conexão

- Socket TCP identificado por 4 valores:
  - Endereço IP de origem
  - End. porta de origem
  - Endereço IP de destino
  - End. porta de destino
- Hospedeiro receptor usa os quatro valores para direccionar o segmento ao socket apropriado
- Hospedeiro servidor pode suportar vários sockets TCP simultâneos:
  - Cada socket é identificado pelos seus próprios 4 valores
  - Servidores Web possuem sockets diferentes para cada cliente conectado
  - HTTP não persistente terá um socket diferente para cada requisição

# 3 Demux orientada à conexão



# 3 Demux orientada à conexão servidor Web “threaded”



# 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 UDP: User Datagram Protocol [RFC 768]

- Protocolo de transporte da Internet “sem gorduras”, “sem frescuras”
- Serviço “best effort”, segmentos UDP podem ser:
  - Perdidos
  - Entregues fora de ordem para a aplicação
- **Sem conexão:**
  - Não há apresentação entre o UDP transmissor e o receptor
  - Cada segmento UDP é tratado de forma independente dos outros

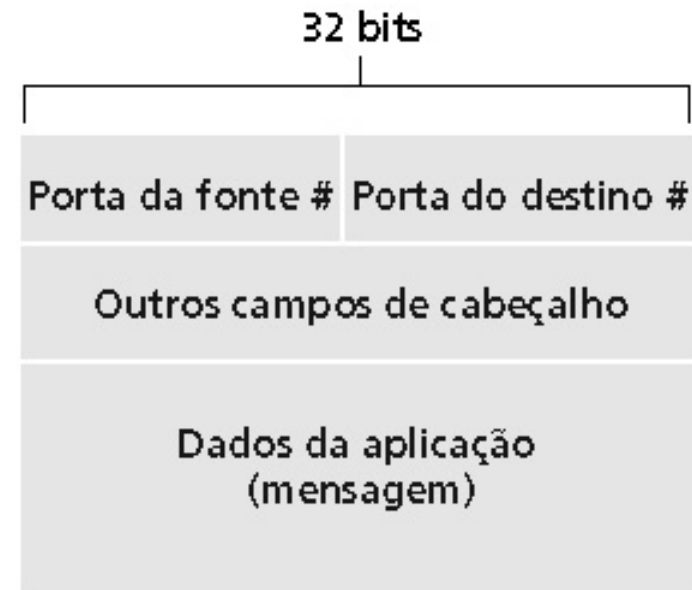
## Por que existe um UDP?

- Não há estabelecimento de conexão (que possa redundar em atrasos)
- Simples: não há estado de conexão nem no transmissor, nem no receptor
- Cabeçalho de segmento reduzido
- Não há controle de congestionamento: UDP pode enviar segmentos tão rápido quanto desejado (e possível)



# 3 Mais sobre UDP

- Muito usado por aplicações de multimídia contínua (streaming)
  - Tolerantes à perda
  - Sensíveis à taxa
- Outros usos do UDP (por quê?):
  - DNS
  - SNMP
- Transferência confiável sobre UDP: acrescentar confiabilidade na camada de aplicação
  - Recuperação de erro específica de cada aplicação



# 3 UDP checksum

**Objetivo:** detectar “erros” (ex.: bits trocados) no segmento transmitido

**Transmissor:**

- Trata o conteúdo do segmento como seqüência de inteiros de 16 bits
- Checksum: soma (complemento de 1 da soma) do conteúdo do segmento
- Transmissor coloca o valor do checksum no campo de checksum do UDP

**Receptor:**

- Computa o checksum do segmento recebido
- Verifica se o checksum calculado é igual ao valor do campo checksum:
  - NÃO - erro detectado
  - SIM - não há erros. **Mas talvez haja erros apesar disso? Mas depois...**

# 3 Exemplo: Internet checksum

- Note que:
  - Ao se adicionar números, um *vai um* do bit mais significativo deve ser acrescentado ao resultado
- Exemplo: adicione dois inteiros de 16 bits

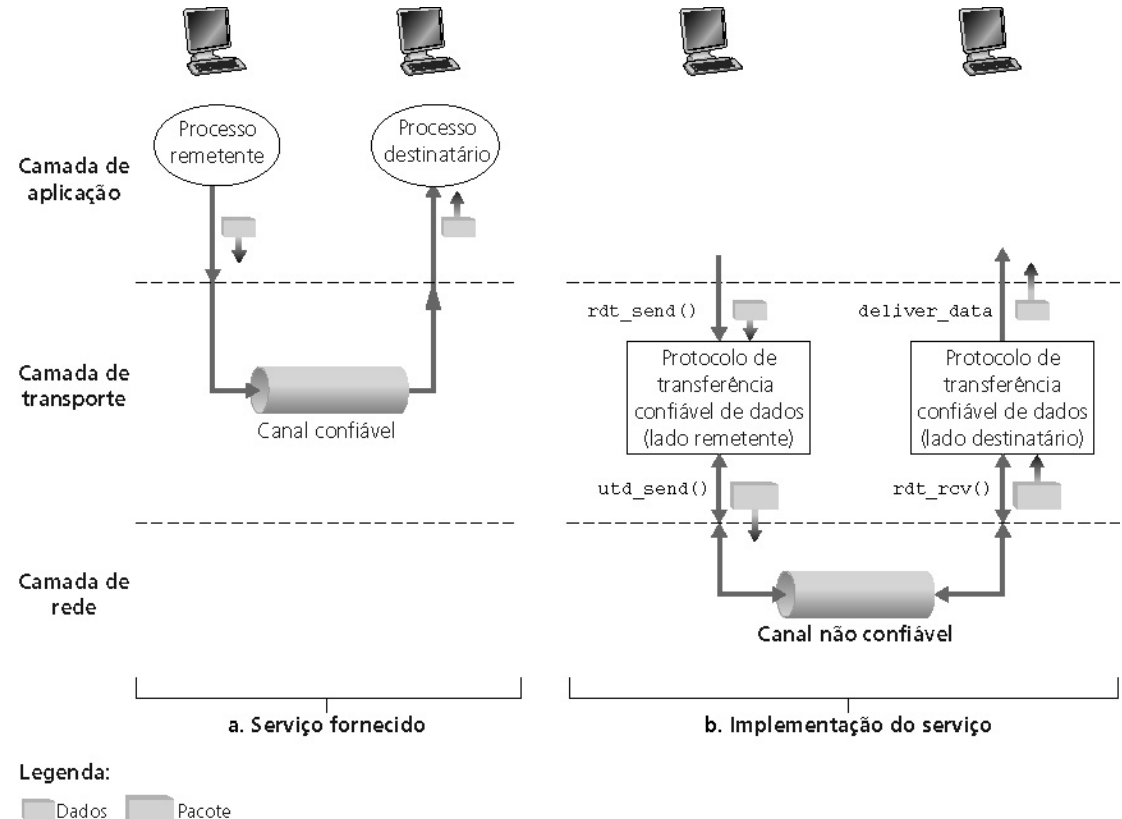
	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
<hr/>																
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

# 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 transferência confiável de dados

- Importante nas camadas de aplicação, transporte e enlace
- Top 10 na lista dos tópicos mais importantes de redes!
- Características dos canais não confiáveis determinarão a complexidade dos protocolos confiáveis de transferência de dados (rdt)

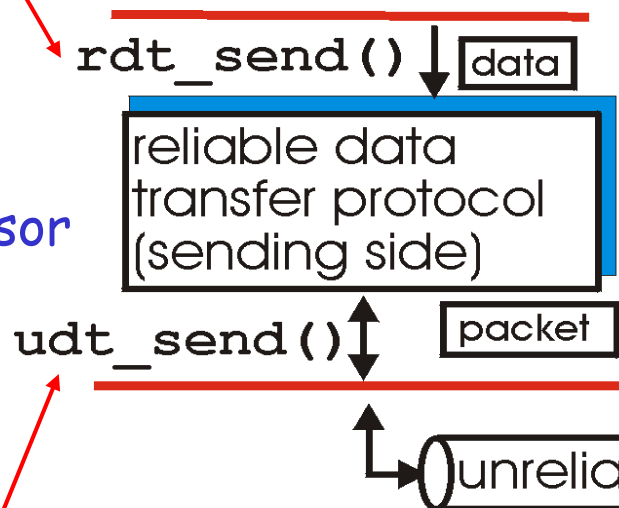


# 3 Transferência confiável: o ponto de partida

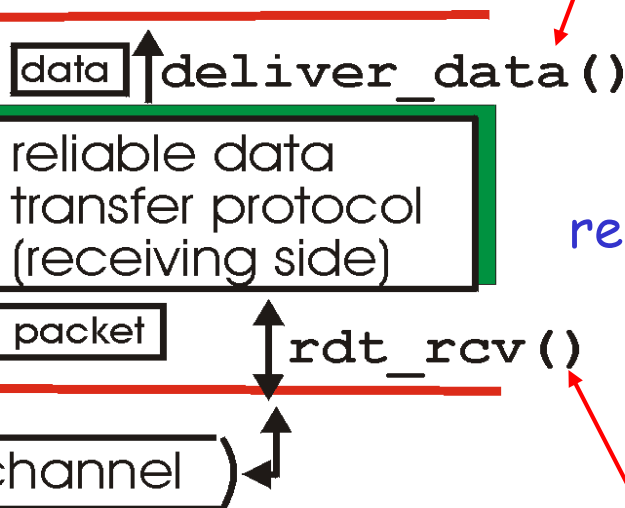
**rdt\_send()** : chamada da camada superior, (ex., pela aplicação). Passa dados para entregar à camada superior receptora

**deliver\_data()** : chamada pela entidade de transporte para entregar dados para cima

lado  
transmissor



**udt\_send()** : chamada pela entidade de transporte, para transferir pacotes para o receptor sobre o canal não confiável



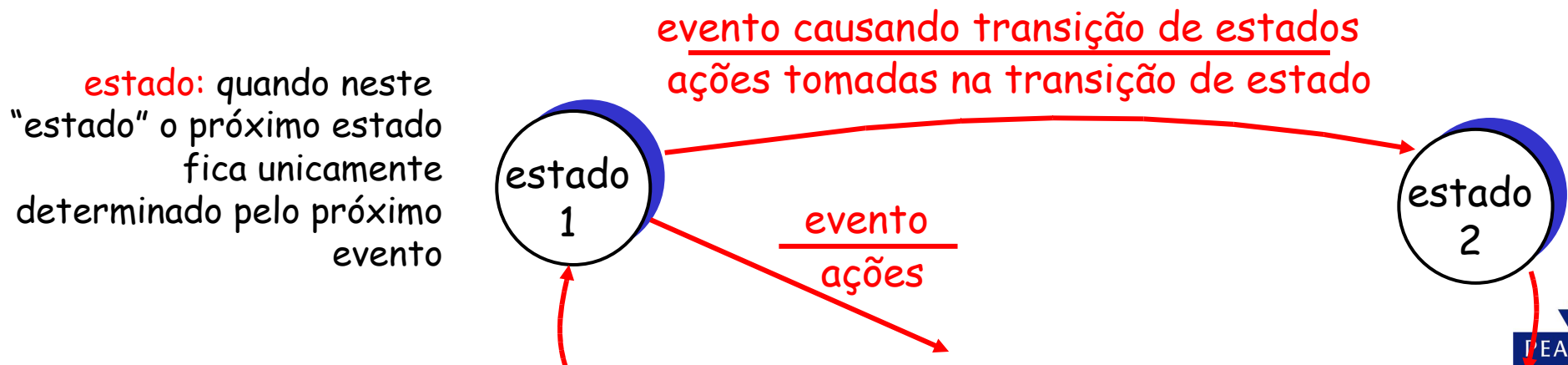
lado  
receptor

**rdt\_rcv()** : chamada quando o pacote chega ao lado receptor do canal

# 3 Transferência confiável: o ponto de partida

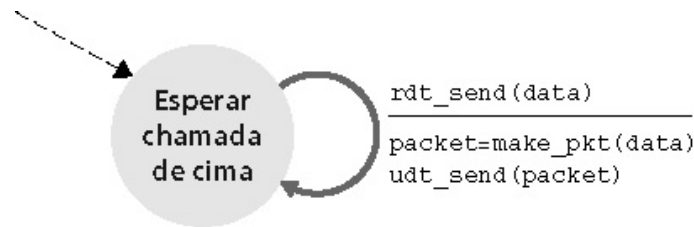
## Etapas:

- Desenvolver incrementalmente o transmissor e o receptor de um protocolo confiável de transferência de dados (rdt)
- Considerar apenas transferências de dados unidirecionais
  - Mas informação de controle deve fluir em ambas as direções!
- Usar máquinas de estados finitos (FSM) para especificar o protocolo transmissor e o receptor

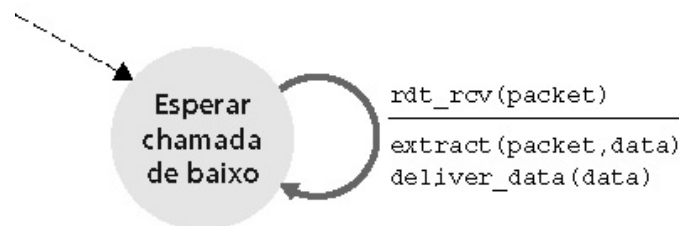


# 3 rdt1.0: Transferência confiável sobre canais confiáveis

- Canal de transmissão perfeitamente confiável
  - Não há erros de bits
  - Não há perdas de pacotes
- FSMs separadas para transmissor e receptor:
  - Transmissor envia dados para o canal subjacente
  - Receptor lê os dados do canal subjacente



a. rdt1.0: lado remetente



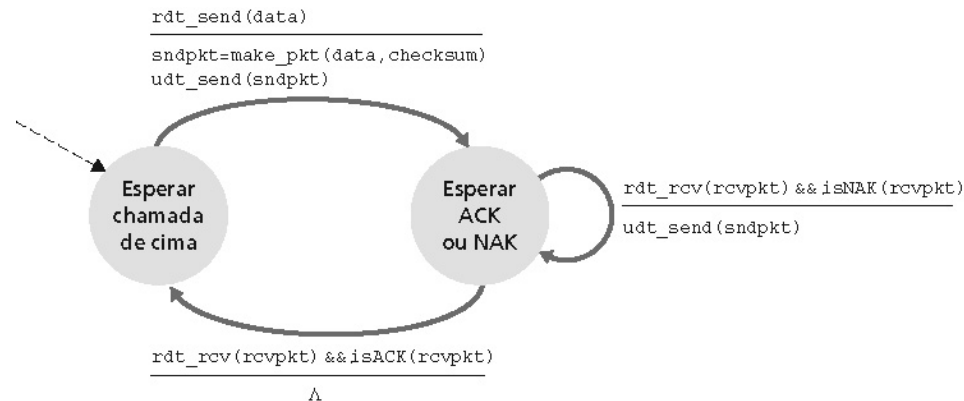
b. rdt1.0: lado destinatário



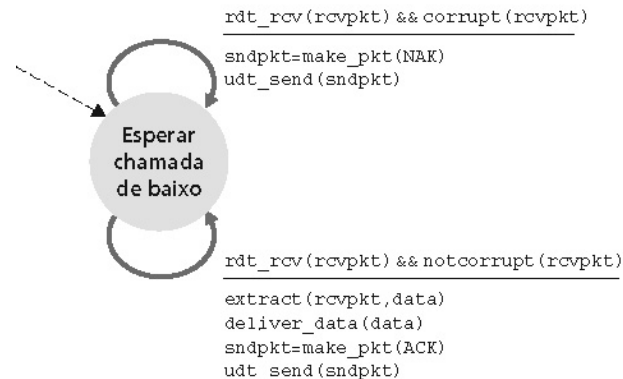
# 3 rdt2.0: canal com erros de bit

- Canal subjacente pode trocar valores dos bits num pacote
  - Checksum para detectar erros de bits
- A questão: como recuperar esses erros:
  - **Reconhecimentos (ACKs)**: receptor avisa explicitamente ao transmissor que o pacote foi recebido corretamente
  - **Reconhecimentos negativos (NAKs)**: receptor avisa explicitamente ao transmissor que o pacote tem erros
  - Transmissor reenvia o pacote quando da recepção de um NAK
- Novos mecanismos no **rdt2.0** (além do **rdt1.0**):
  - Detecção de erros
  - Retorno do receptor: mensagens de controle (ACK, NAK) rcvr->sender

# 3 rdt2.0: especificação FS M

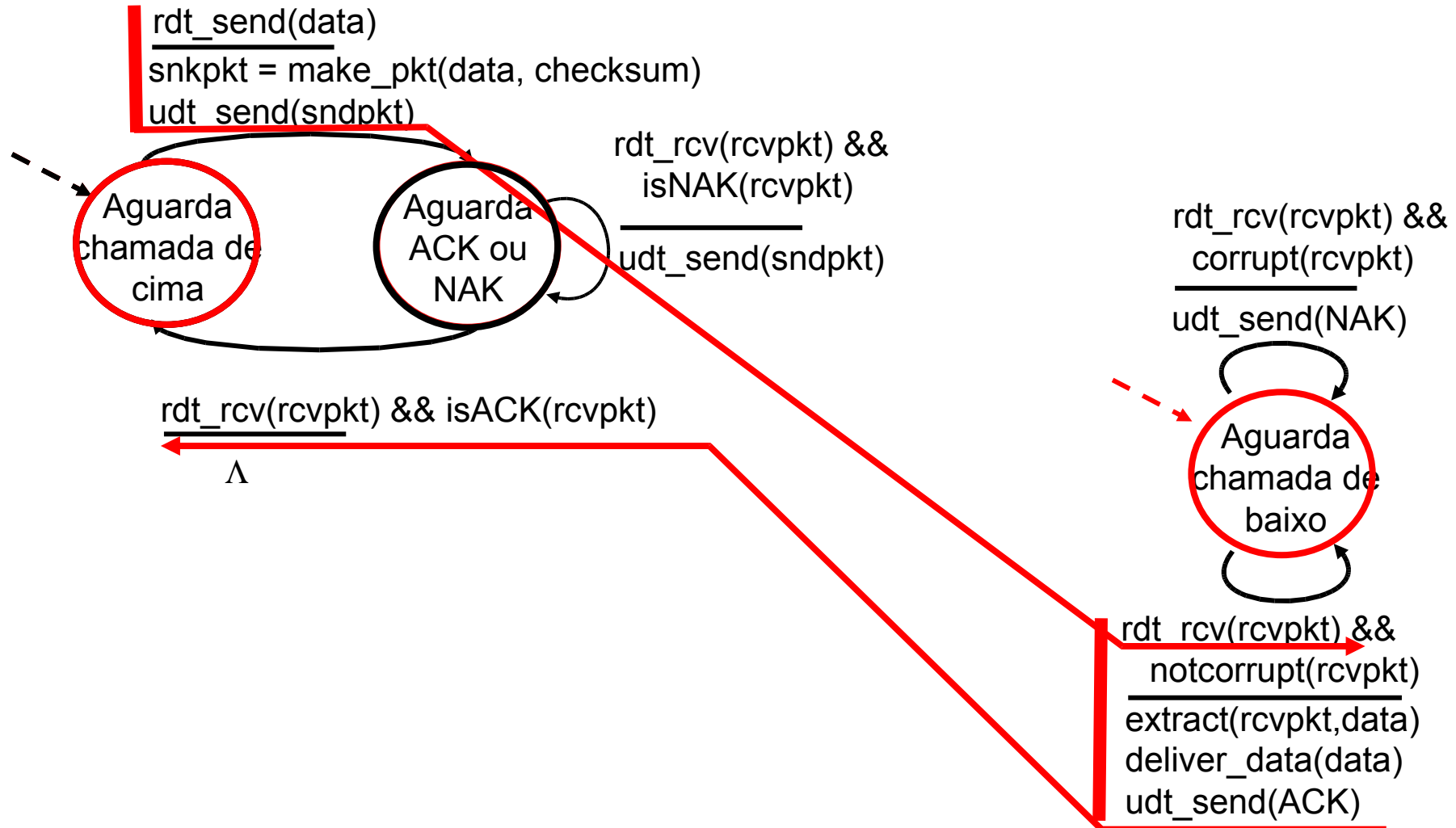


a. rdt2.0: lado remetente

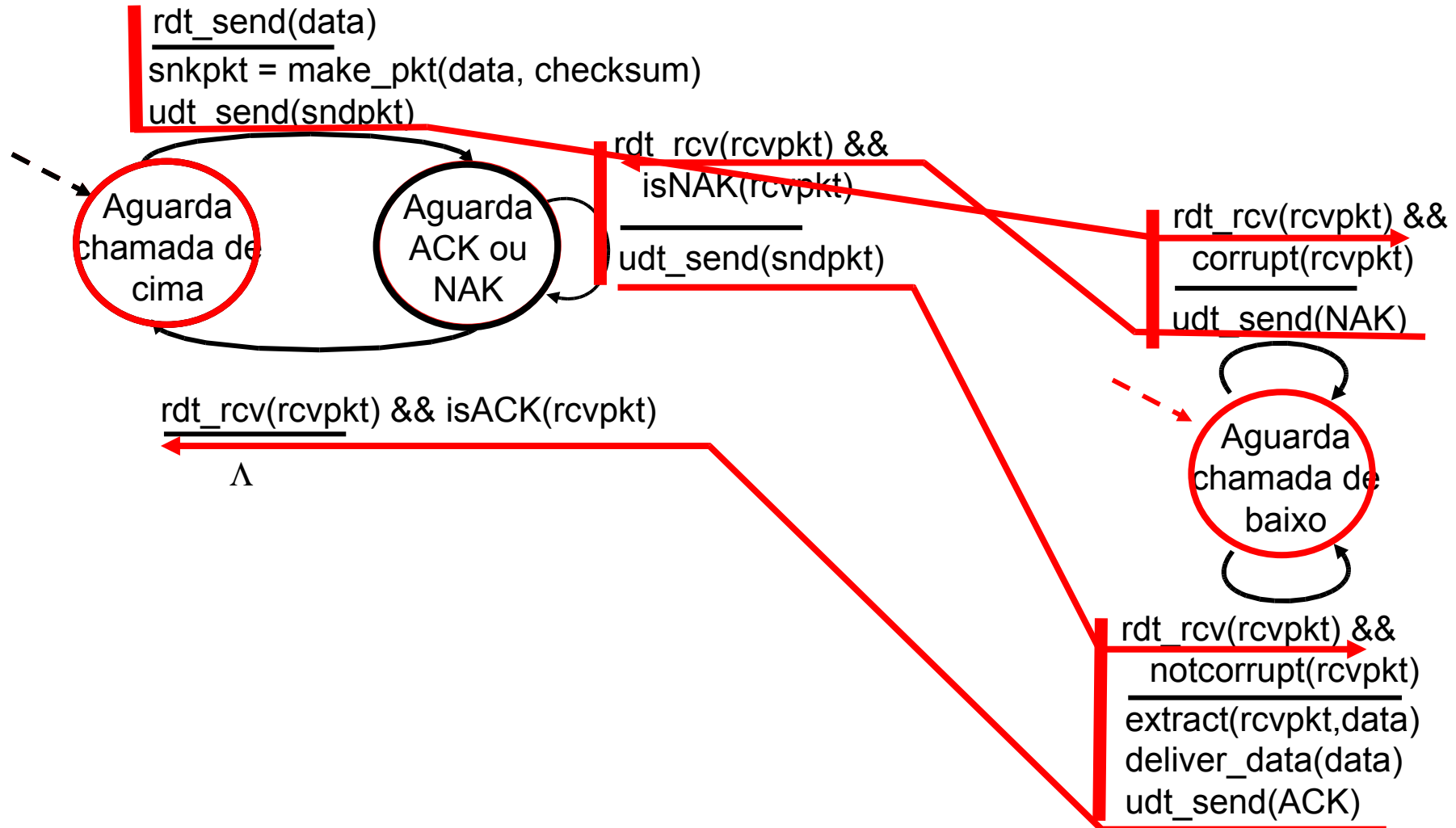


b. rdt2.0: lado destinatário

# 3 rdt2.0 operação com ausência de erros



# 3 rdt2.0: cenário de erro



# 3 rdt2.0 tem um problema fatal!

## O que acontece se o ACK/NAK é corrompido?

- Transmissor não sabe o que aconteceu no receptor!
- Não pode apenas retransmitir: possível duplicata

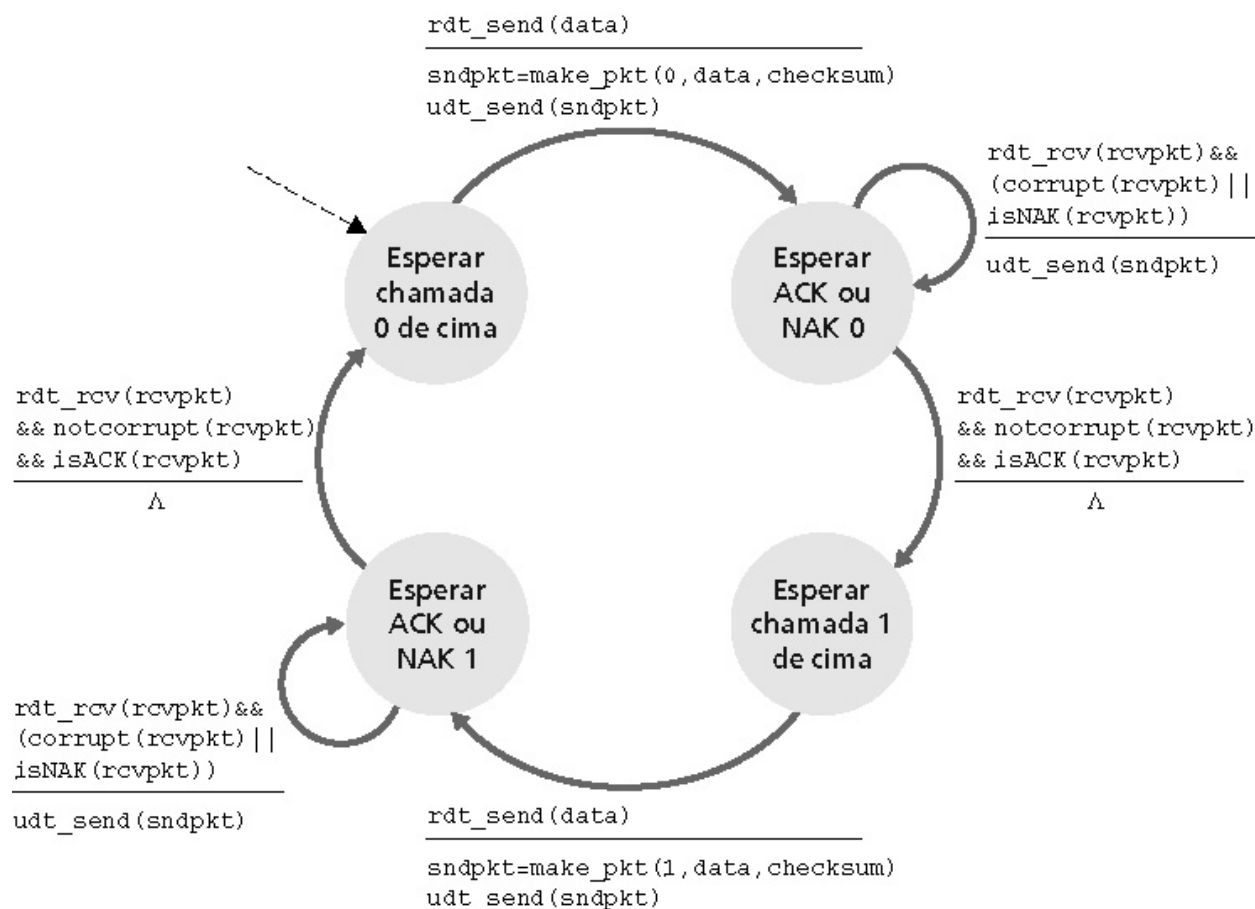
## Tratando duplicatas:

- Transmissor acrescenta **número de seqüência** em cada pacote
- Transmissor reenvia o último pacote se ACK/NAK for perdido
- Receptor descarta (não passa para a aplicação) pacotes duplicados

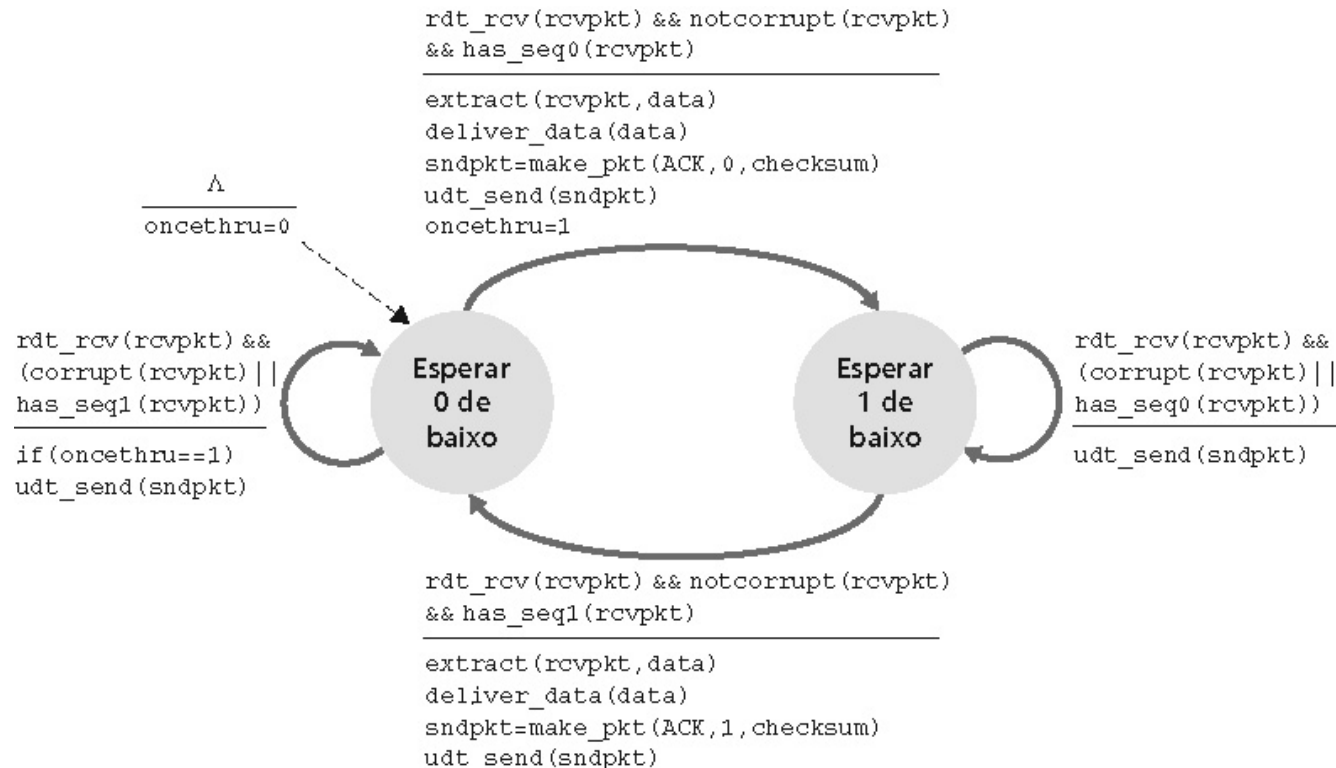
## Pare e espere

Transmissor envia um pacote e então espera pela resposta do receptor

# 3 rdt2.1: transmissor, trata ACK NAKs perdidos



# 3 rdt2.1: receptor, trata ACK NAKs perdidos



# 3 rdt2.1: discussão

## Transmissor:

- Adiciona número de seqüência ao pacote
- Dois números (0 e 1) bastam. Por quê?
- Deve verificar se os ACK/NAK recebidos estão corrompidos
- Duas vezes o número de estados
  - O estado deve “lembrar” se o pacote “corrente” tem número de seqüência 0 ou 1

## Receptor:

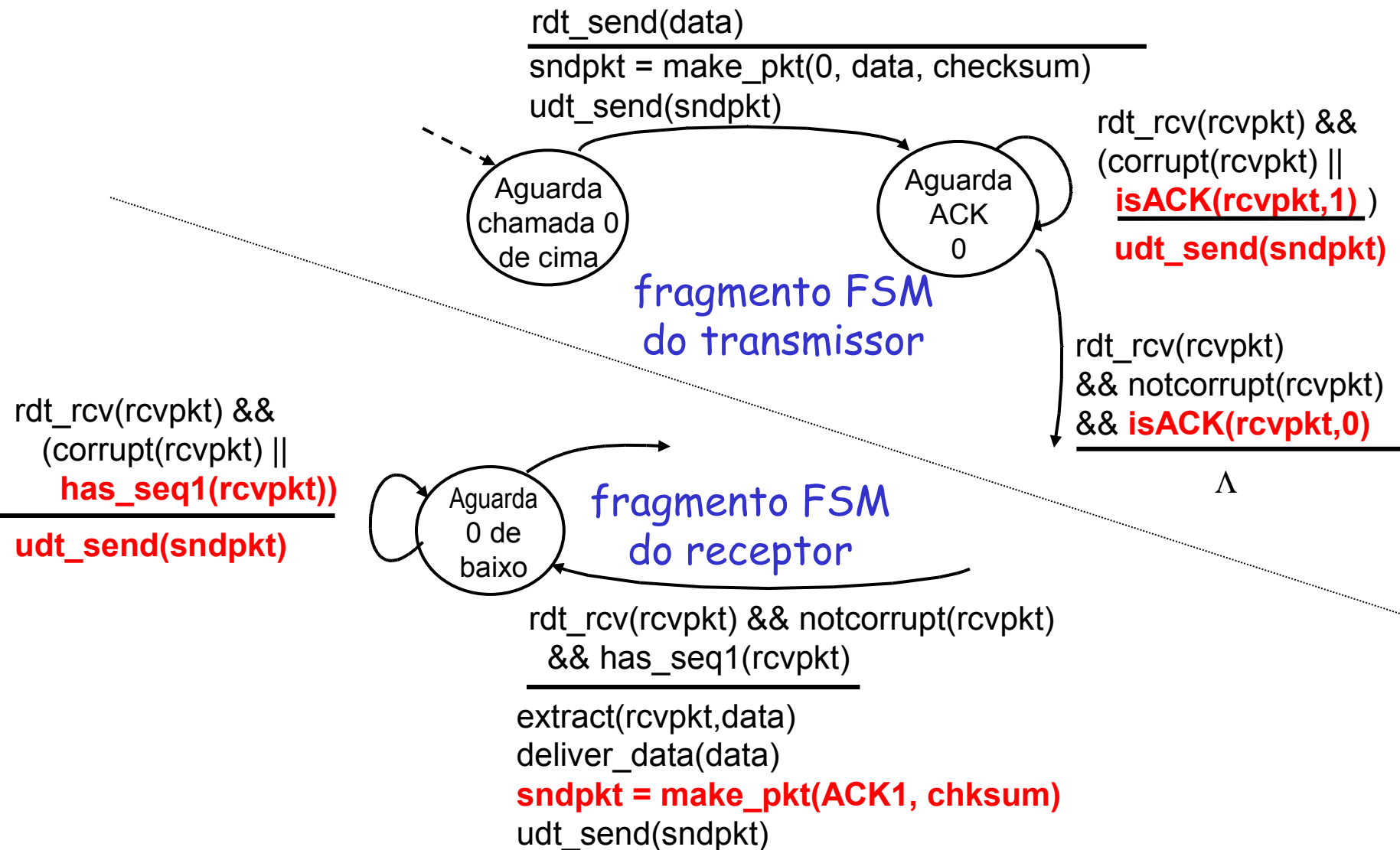
- Deve verificar se o pacote recebido é duplicado
  - Estado indica se o pacote 0 ou 1 é esperado
- Nota: receptor pode não saber se seu último ACK/NAK foi recebido pelo transmissor



# 3 rdt2.2: um protocolo sem NAK

- Mesma funcionalidade do rdt2.1, usando somente ACKs
- Em vez de enviar NAK, o receptor envia ACK para o último pacote recebido sem erro
  - Receptor deve incluir explicitamente o número de seqüência do pacote sendo reconhecido
- ACKs duplicados no transmissor resultam na mesma ação do NAK: **retransmissão do pacote corrente**

# 3 rdt2.2: fragmentos do transmissor e do receptor



# 3 rdt3.0: canais com erros e perdas

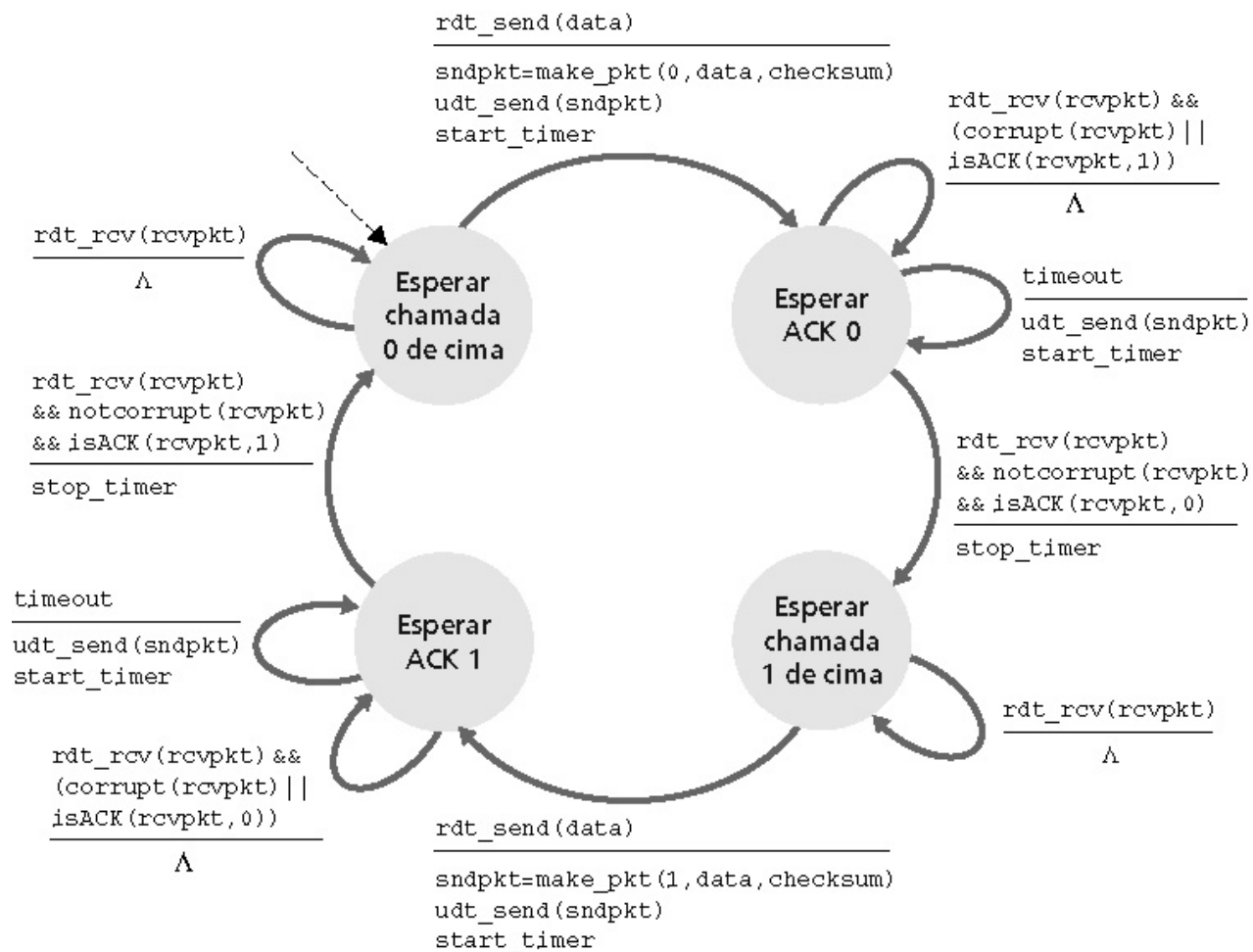
**Nova hipótese:** canal de transmissão pode também perder pacotes (devido aos ACKs)

- Checksum, números de seqüência, ACKs, retransmissões serão de ajuda, mas não o bastante

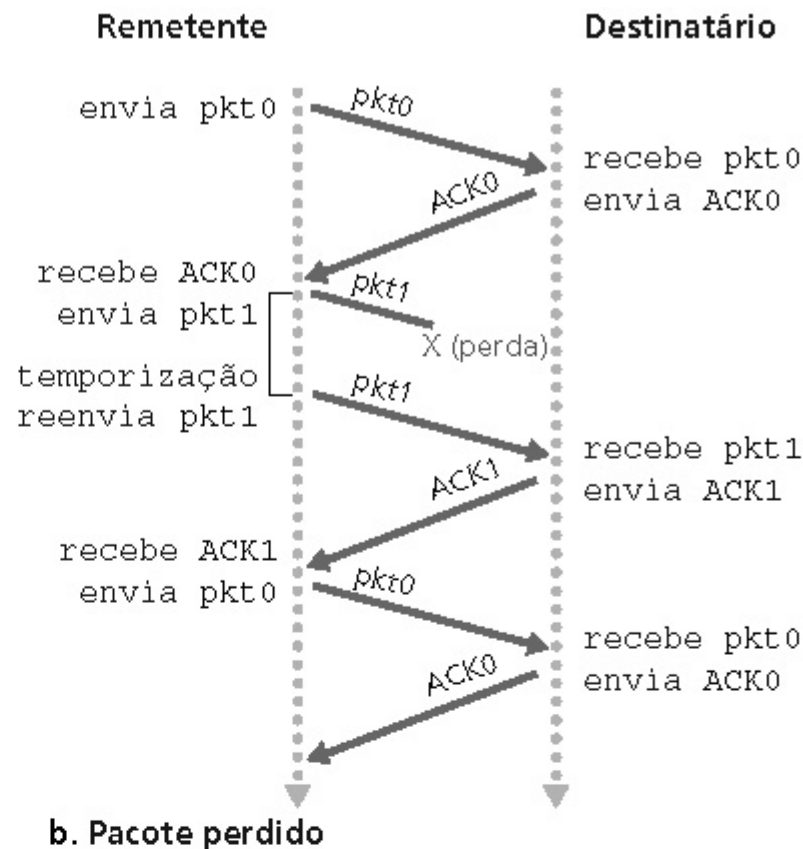
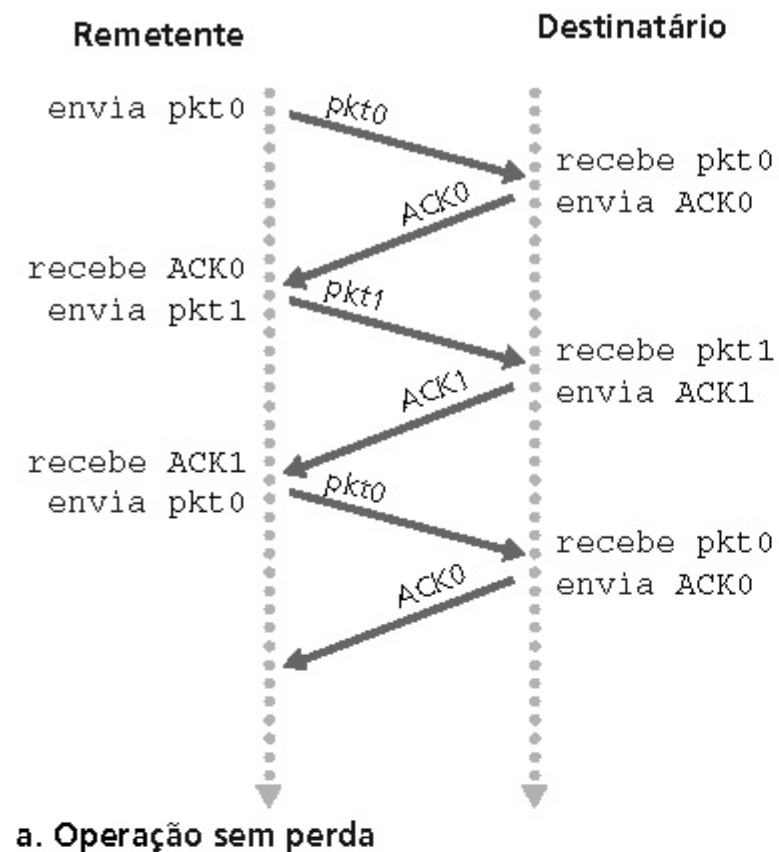
**Abordagem:** transmissor espera um tempo “razoável” pelo ACK

- Retransmite se nenhum ACK for recebido nesse tempo
- Se o pacote (ou ACK) estiver apenas atrasado (não perdido):
- Retransmissão será duplicata, mas os números de seqüência já tratam com isso
- Receptor deve especificar o número de seqüência do pacote sendo reconhecido
- Exige um temporizador decrescente

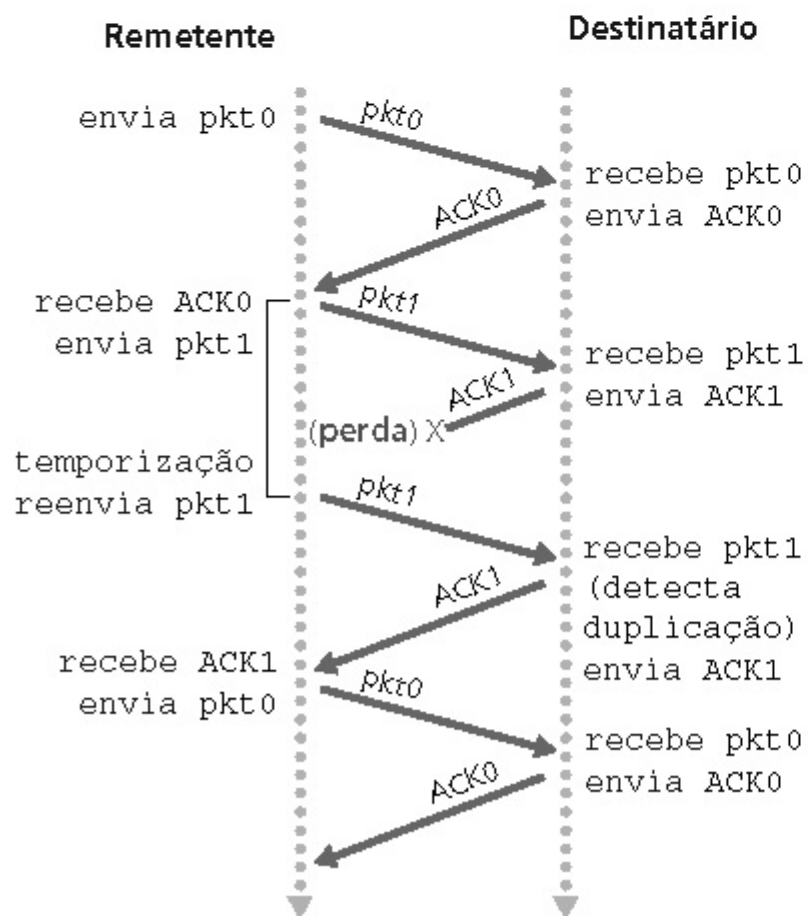
# 3 Transmissor rdt3.0



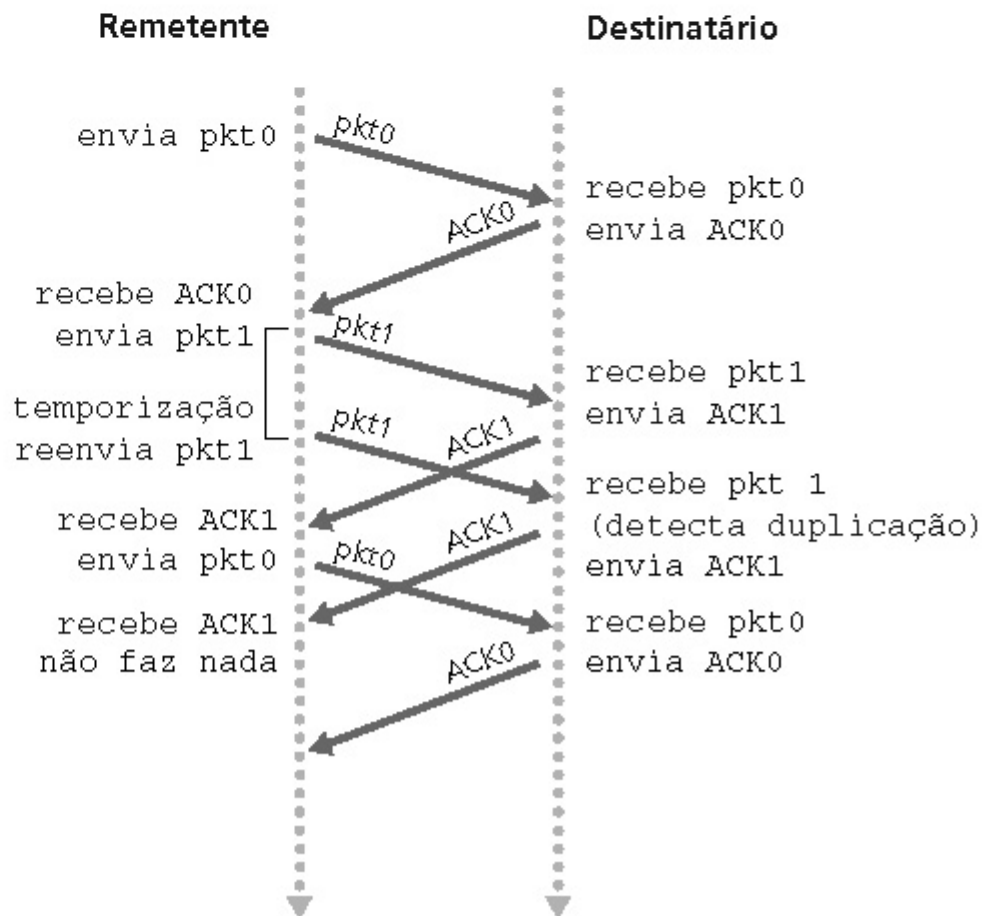
# 3 rdt3.0 em ação



# 3 rdt3.0 em ação



c. ACK perdido



d. Interrupção prematura

# 3 Desempenho do rdt3.0

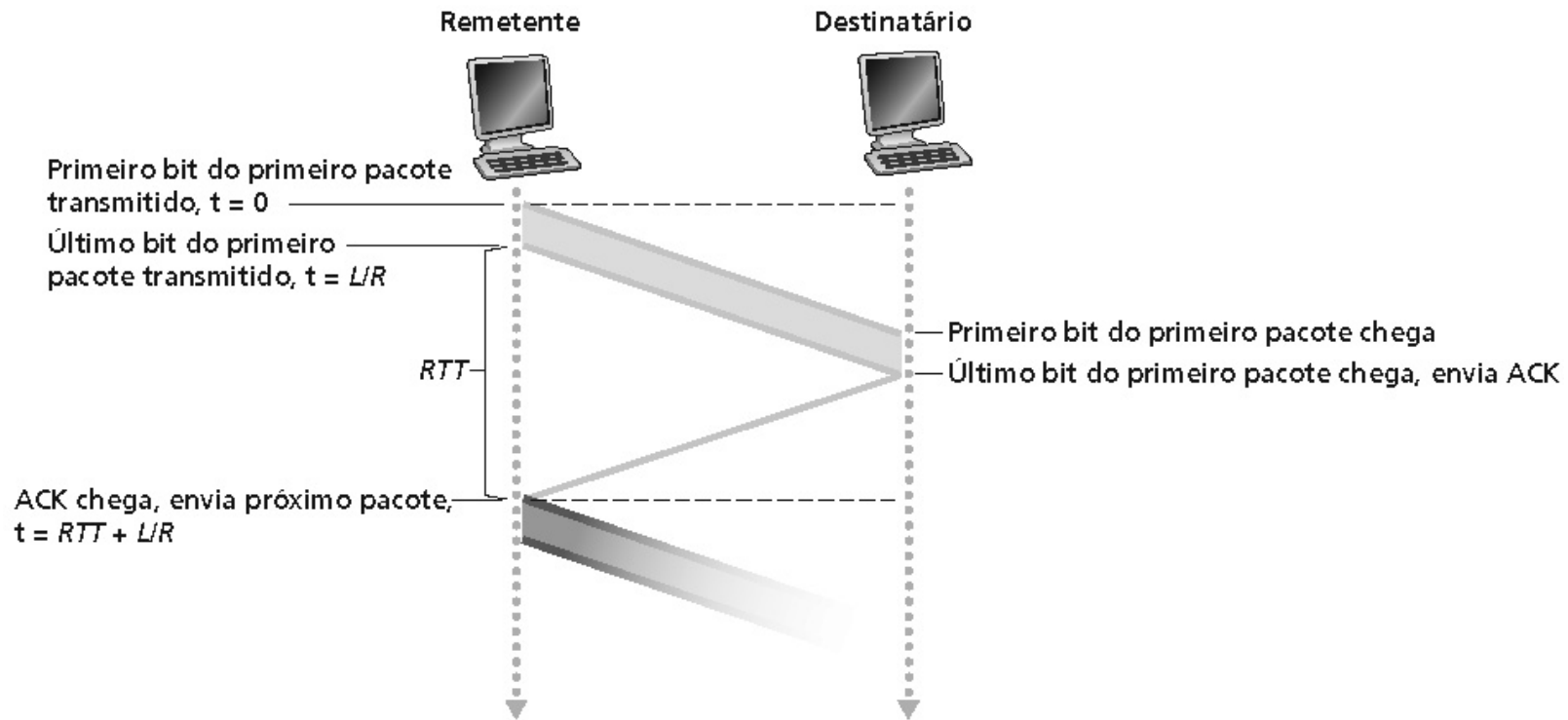
- rdt3.0 funciona, mas o desempenho é sofrível
- Exemplo: enlace de 1 Gbps, 15 ms de atraso de propagação, pacotes de 1 KB:

$$\text{Transmissão} = \frac{L \text{ (tamanho do pacote em bits)}}{R \text{ (taxa de transmissão, bps)}} = \frac{8 \text{ kb/pkt}}{10^{**9} \text{ b/s}} = 8 \text{ microsseg}$$

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$

- $U_{\text{sender}}$ : **utilização** — fração de tempo do transmissor ocupado
- Um pacote de 1 KB cada 30 ms -> 33 kB/s de vazão sobre um canal
- De 1 Gbps
- O protocolo de rede limita o uso dos recursos físicos!

# 3 rdt3.0: operação *pare e espere*



a. Operação *pare e espere*

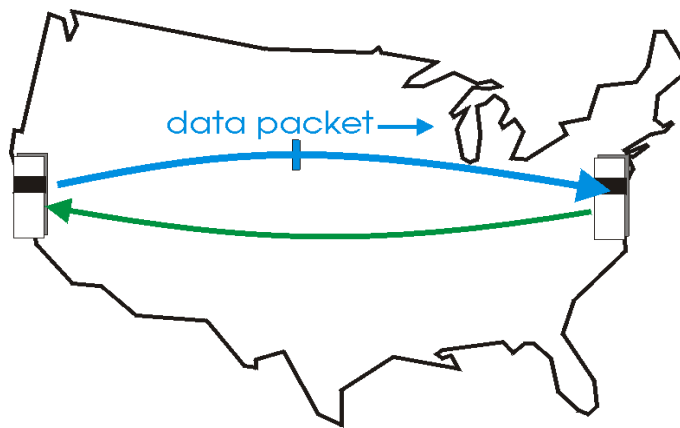
$$\text{sender} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$



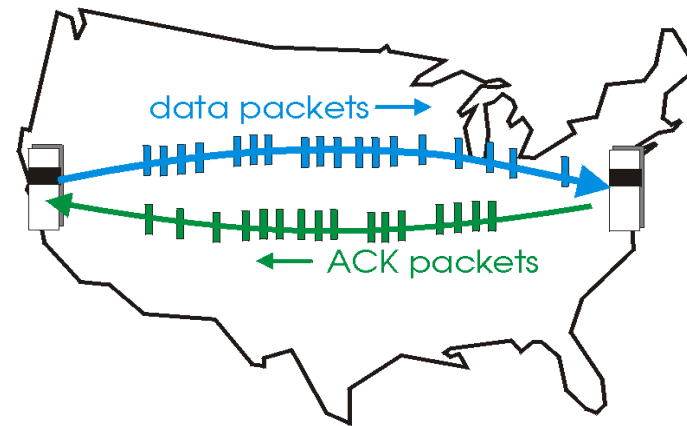
# 3 Protocolos com paralelismo (pipelining)

**Paralelismo:** transmissor envia vários pacotes ao mesmo tempo, todos esperando para serem reconhecidos

- Faixa de números de seqüência deve ser aumentada
- Armazenamento no transmissor e/ou no receptor



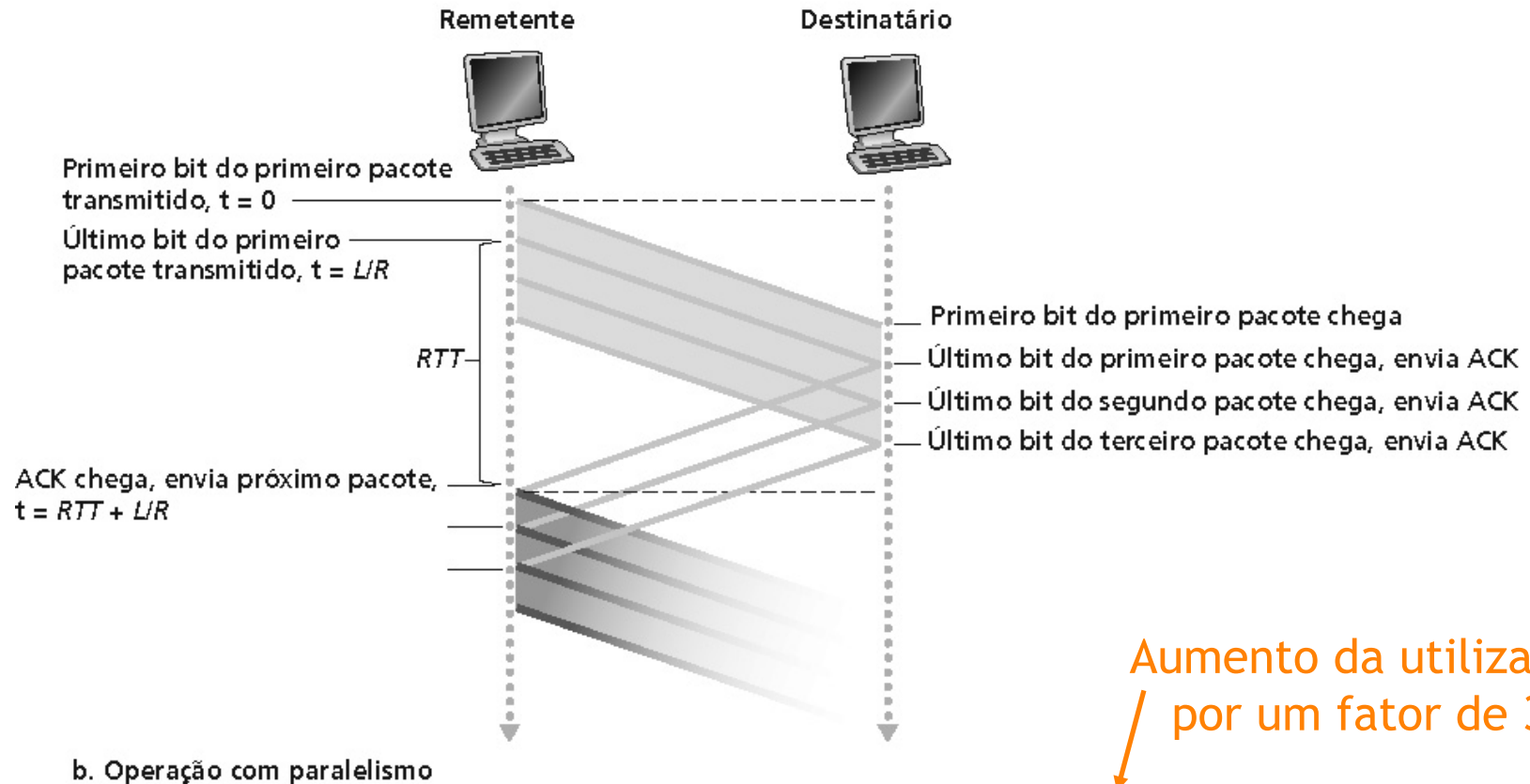
(a) operação do protocolo pare e espere



(a) operação do protocolo com paralelismo

- Duas formas genéricas de protocolos com paralelismo: **go-Back-N**, **retransmissão seletiva**

# 3 Pipelining: aumento da utilização

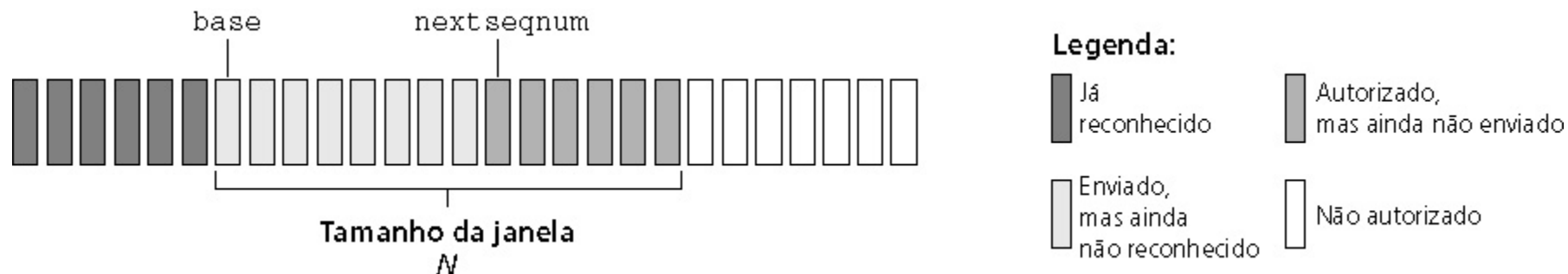


$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L / R} = \frac{0,024}{30,008} = 0,0008$$

# 3 Go-Back-N

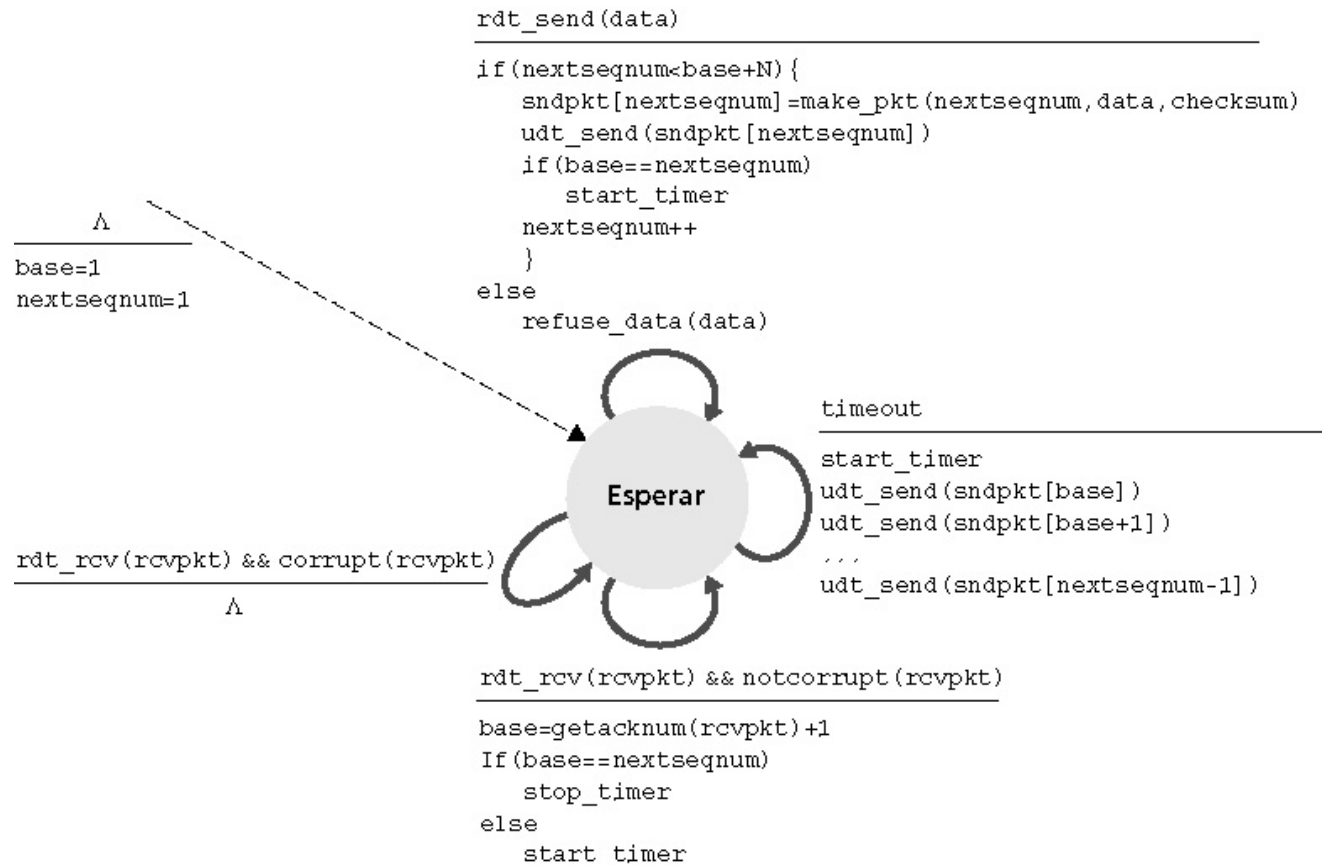
## Transmissor:

- Número de seqüência com k bits no cabeçalho do pacote
- “janela” de até N pacotes não reconhecidos, consecutivos, são permitidos

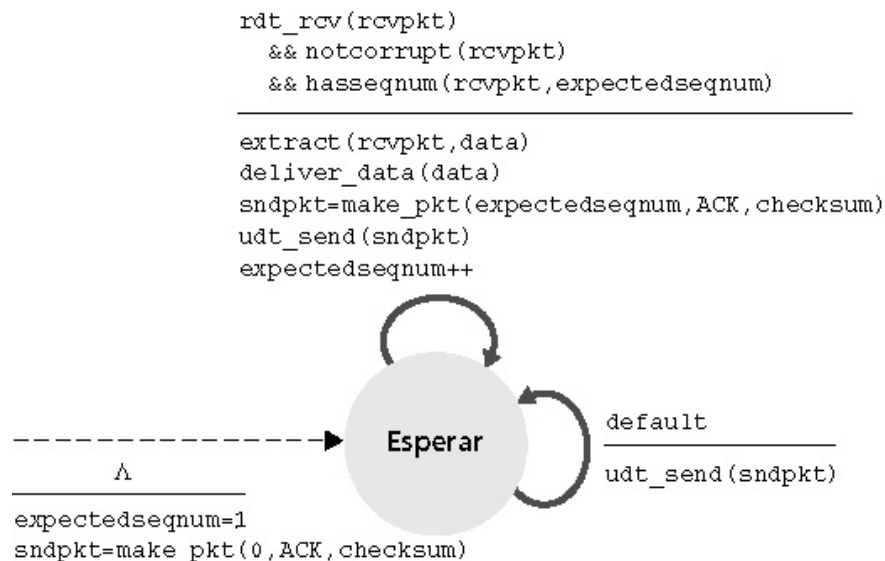


- ACK(n): reconhece todos os pacotes até o número de seqüência N (incluindo este limite). “ACK cumulativo”
  - Pode receber ACKs duplicados (veja receptor)
- Temporizador para cada pacote enviado e não confirmado
- **Tempo de confirmação (n):** retransmite pacote n e todos os pacotes com número de seqüência maior que estejam dentro da janela

# 3 GBN: FSM estendida para o transmissor

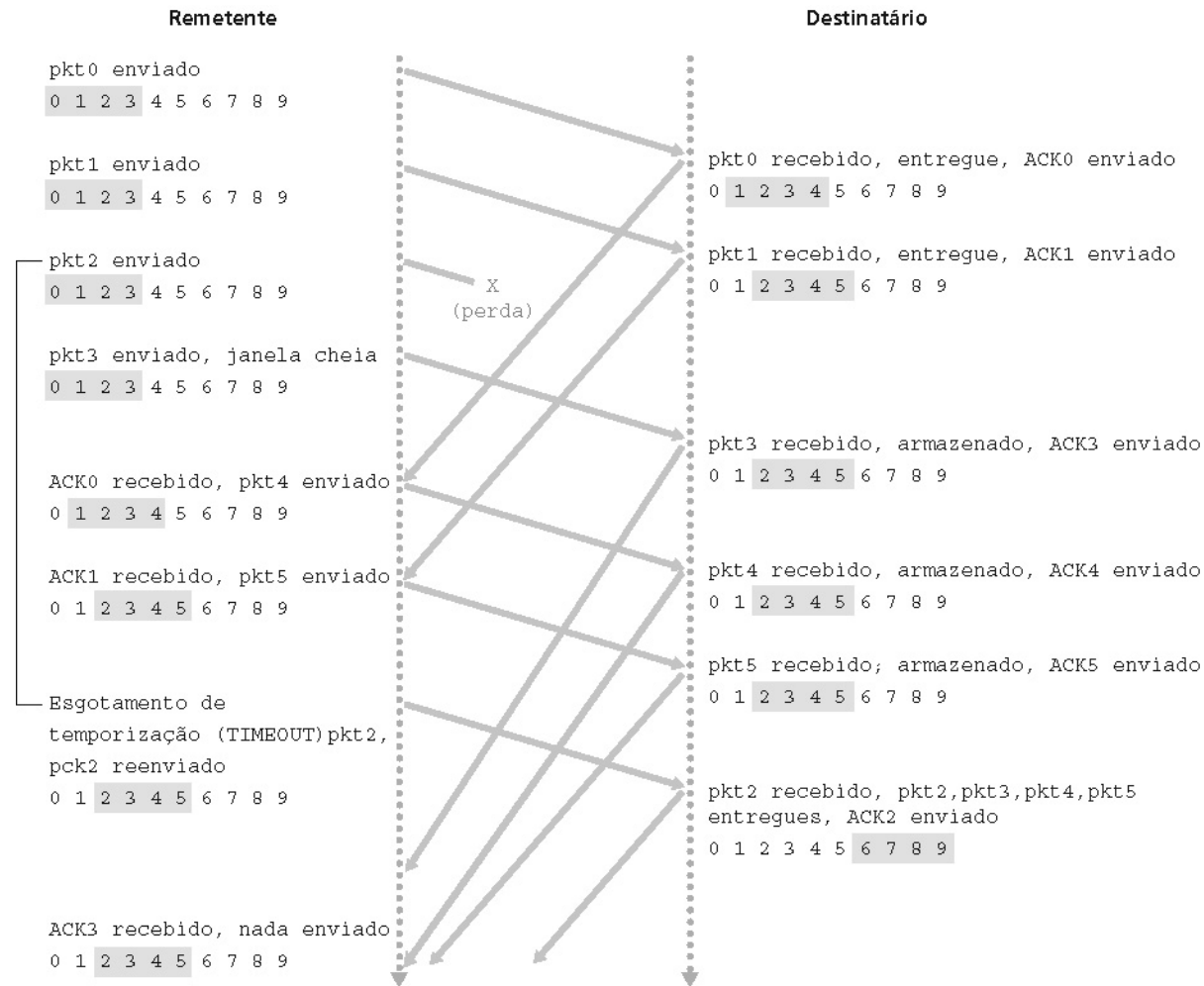


# 3 GBN: FSM estendida para o receptor



- Somente ACK: sempre envia ACK para pacotes corretamente recebidos com o mais alto número de seqüência **em ordem**
  - Pode gerar ACKs duplicados
  - Precisa lembrar apenas do **expectedseqnum**
- Pacotes fora de ordem:
  - Descarta (não armazena) -> **não há buffer de recepção!**
  - Reconhece pacote com o mais alto número de seqüência em ordem

# 3 GBN em ação

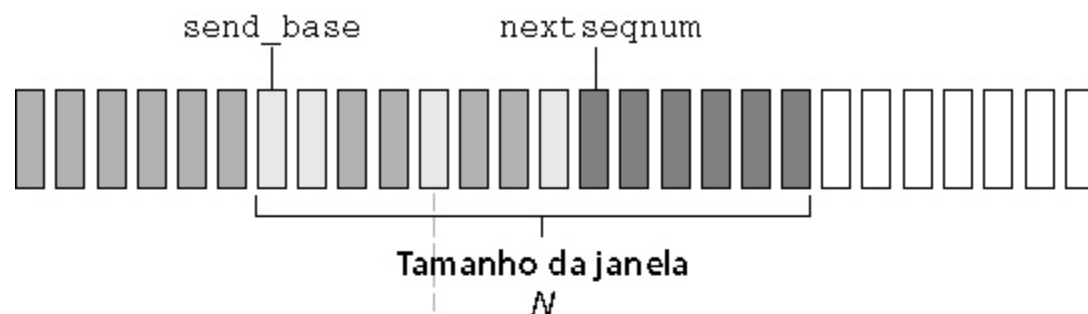


# 3 Retransmissão seletiva

- Receptor reconhece **individualmente** todos os pacotes recebidos corretamente
  - Armazena pacotes, quando necessário, para eventual entrega em ordem para a camada superior
- Transmissor somente reenvia os pacotes para os quais um ACK não foi recebido
  - Transmissor temporiza cada pacote não reconhecido
- Janela de transmissão
  - N números de sequência consecutivos
  - Novamente limita a quantidade de pacotes enviados, mas não reconhecidos

# 3

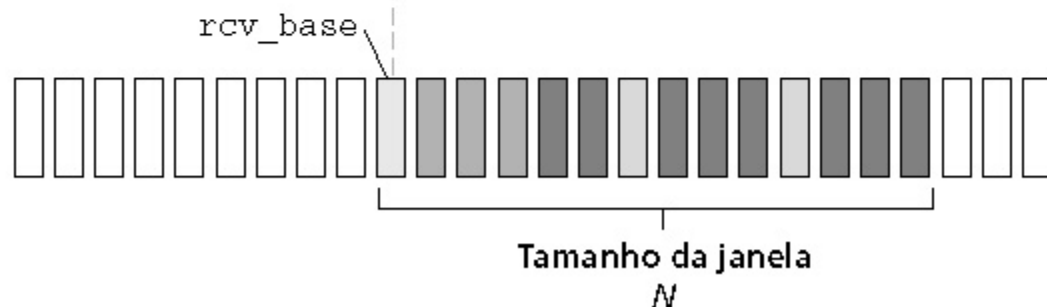
## Retransmissão seletiva: janelas do transmissor e do receptor



a. Visão que o remetente tem dos números de sequência

### Legenda:

	Já reconhecido		Autorizado, mas ainda não enviado
	Enviado, mas não autorizado		Não autorizado



b. Visão que o destinatário tem dos números de sequência

### Legenda

	Fora de ordem (no buffer), mas já reconhecido (ACK)		Aceitável (dentro da janela)
	Aguardado, mas ainda não recebido		Não autorizado





# 3 Retransmissão seletiva

## TRANSMISSOR

### Dados da camada superior:

- Se o próximo número de seqüência disponível está na janela, envia o pacote

### Tempo de confirmação(n):

- Reenvia pacote n, restart timer

### ACK (n) em [sendbase,sendbase+N]:

- Marca pacote n como recebido
- Se n é o menor pacote não reconhecido, avança a base da janela para o próximo número de seqüência não reconhecido

## RECEPTOR

### Pacote n em [rcvbase, rcvbase + N -1]

- Envia ACK(n)
- Fora de ordem: armazena
- Em ordem: entrega (também entrega pacotes armazenados em ordem), avança janela para o próximo pacote ainda não recebido

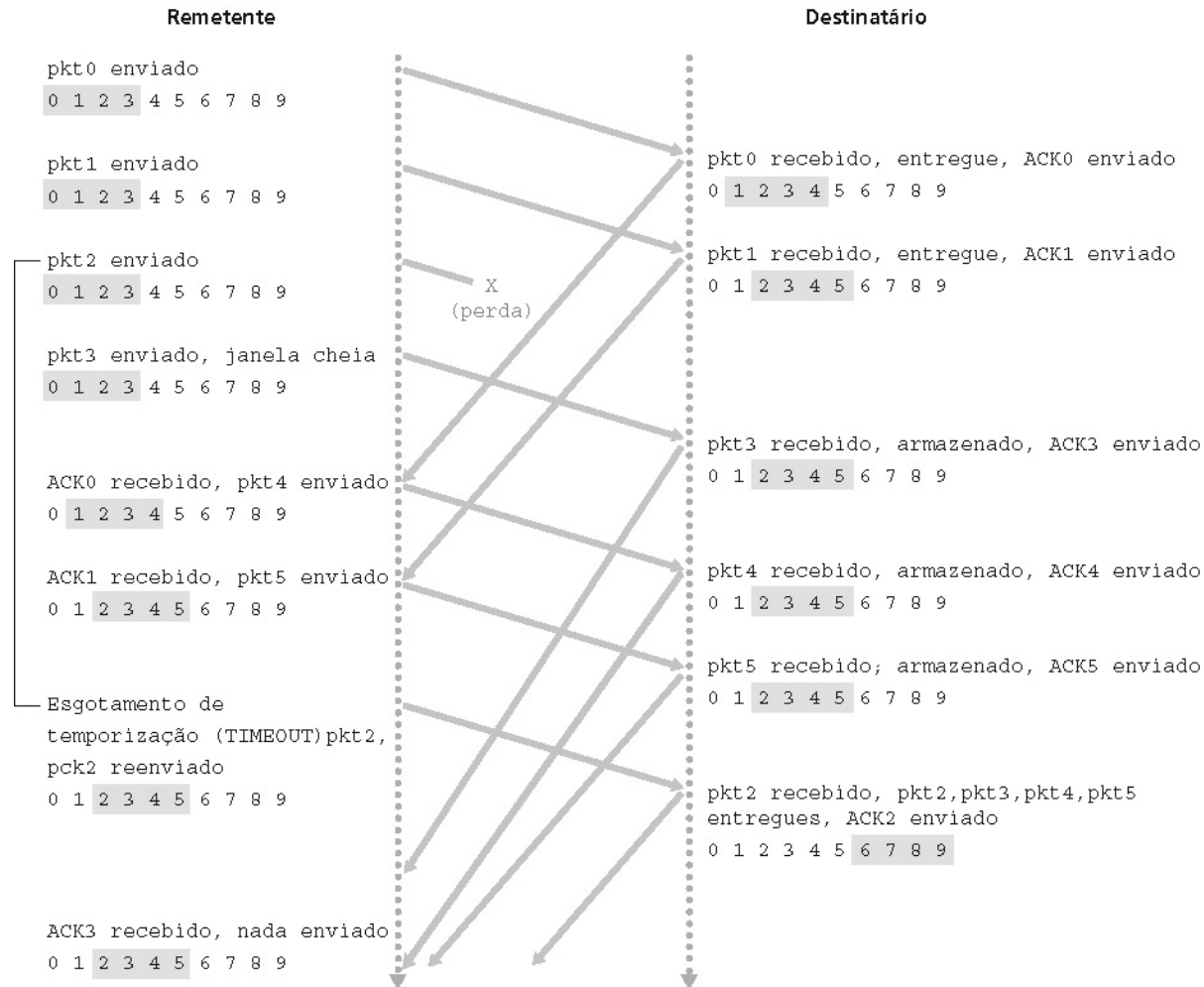
### pkt n em [rcvbase-N,rcvbase-1]

- ACK(n)

### Caso contrário:

- Ignora

# 3 Retransmissão seletiva em ação

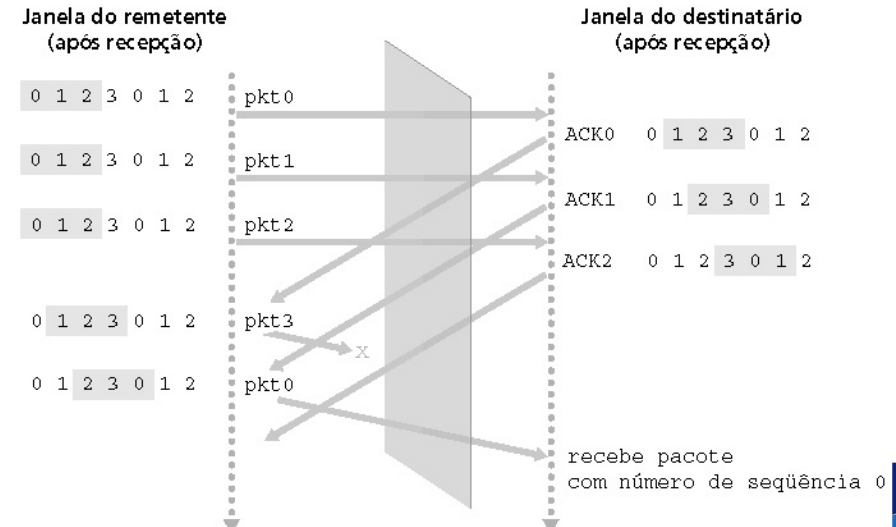
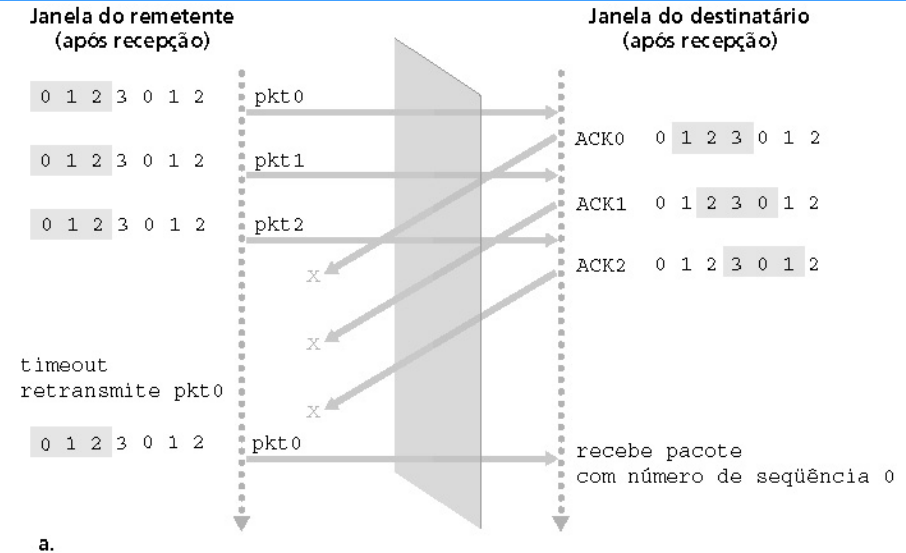


# 3 Retransmissão seletiva: dilema

Exemplo:

- Sequências: 0, 1, 2, 3
- Tamanho da janela = 3
- Receptor não vê diferença nos dois cenários!
- Incorretamente passa dados duplicados como novos (figura a)

P.: Qual a relação entre o espaço de numeração seqüencial e o tamanho da janela?



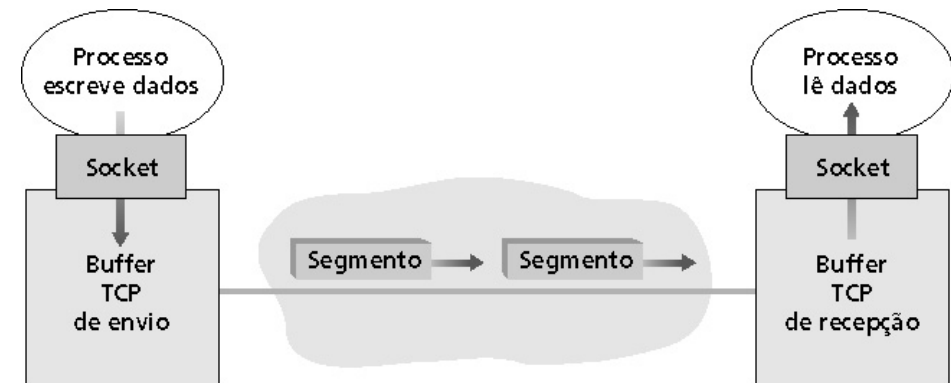
# 3 Camada de transporte

- 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

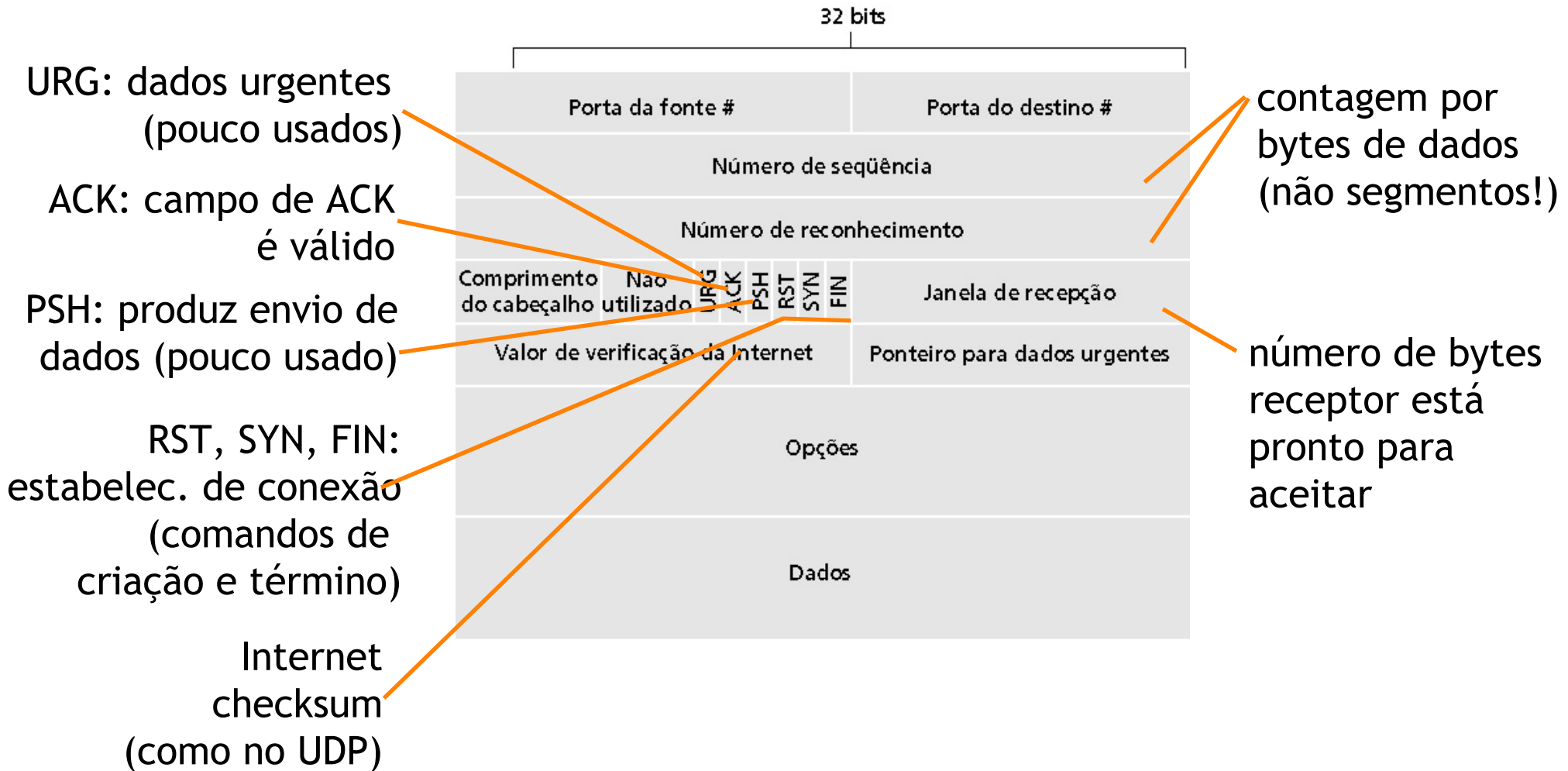
# 3 TCP: overview

RFCs: 793, 1122, 1323, 2018, 2581

- **Ponto-a-ponto:**
  - Um transmissor, um receptor
- **Confiável, sequencial 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



# 3 Estrutura do segmento TCP



# 3 Número de seqüência e ACKs do TCP

## Números de seqüência:

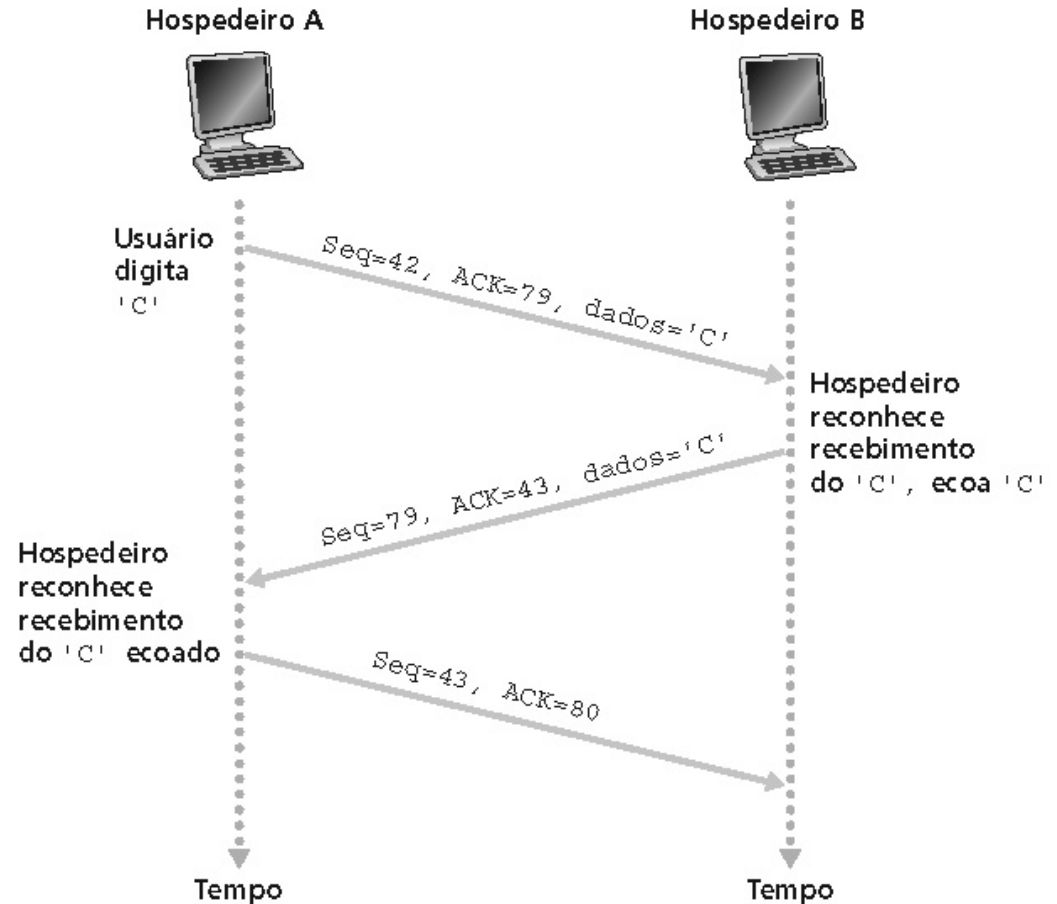
- Número do primeiro byte nos segmentos de dados

## 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



# 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



PEARSON

Addison  
Wesley

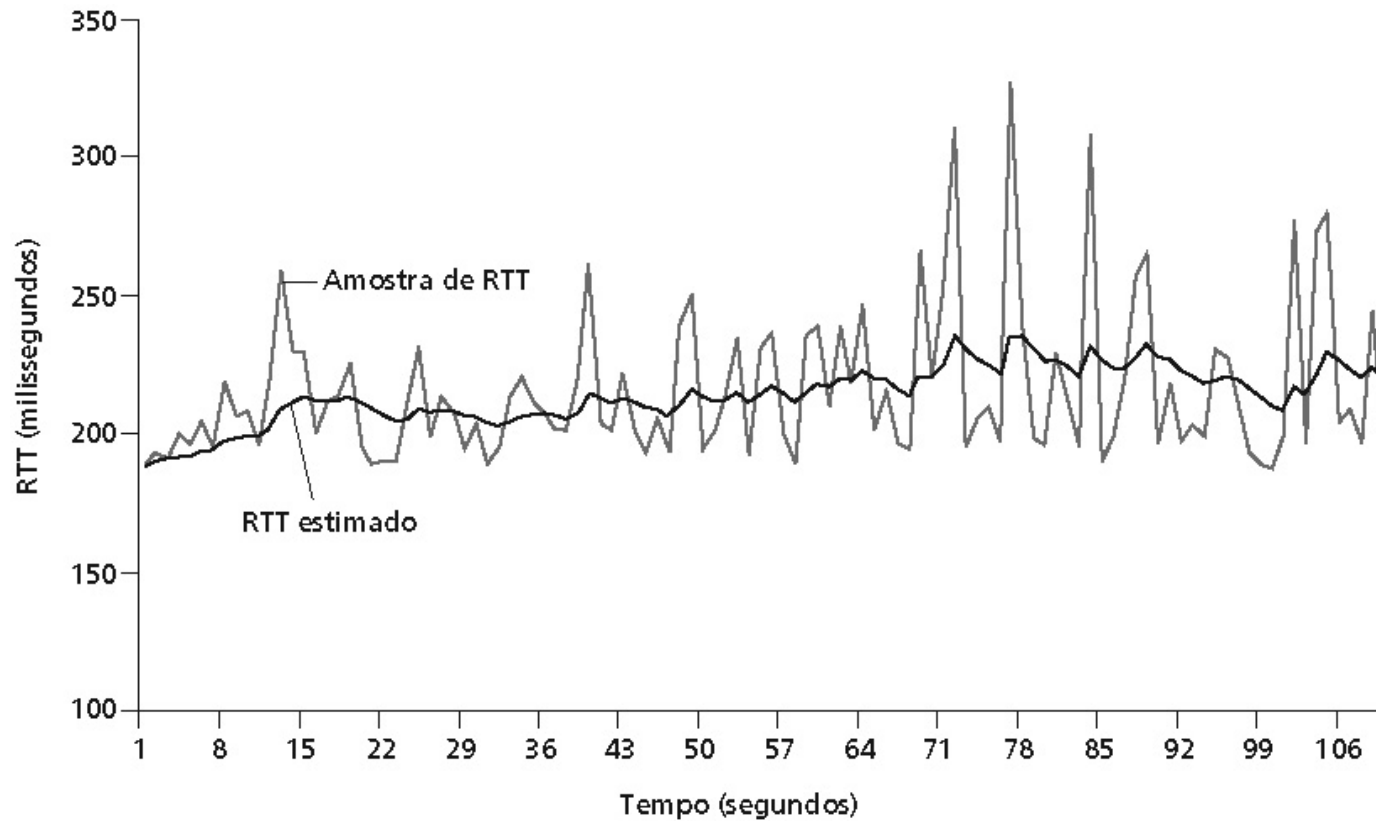


# 3 TCP Round Trip Time e temporização

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

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

# 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**:

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

(typically,  $\beta = 0.25$ )

## Então ajustar o intervalo de temporização

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

# 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:**

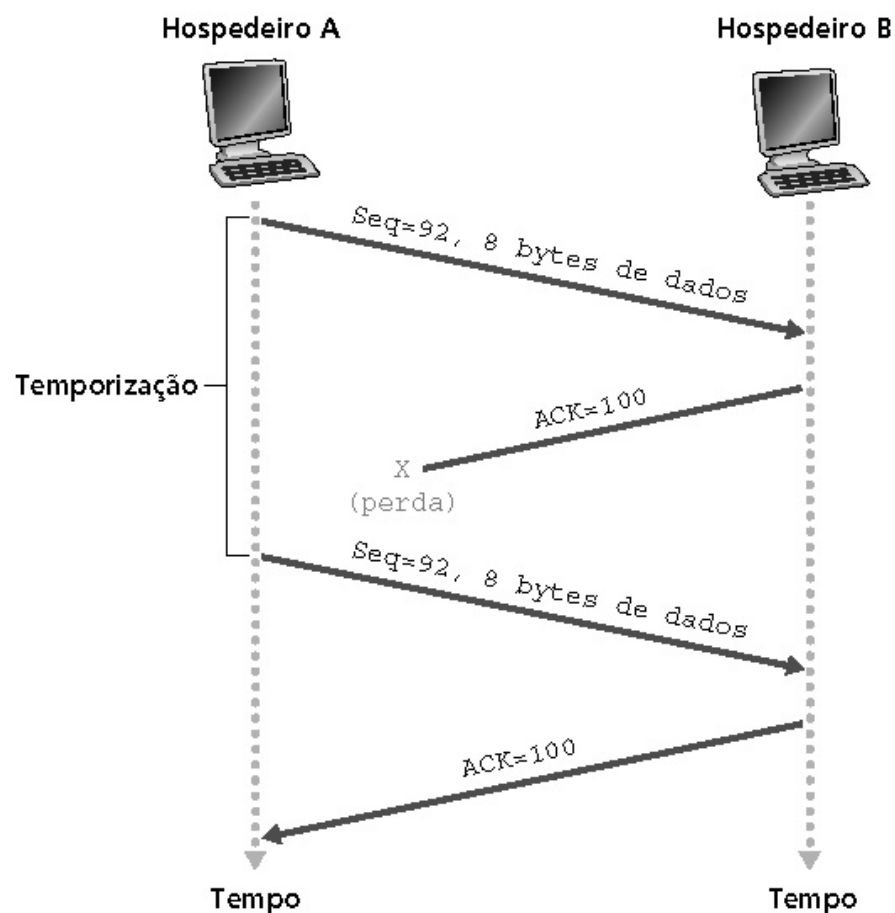
- SendBase-1:  
último byte do  
ACK cumulativo

**Exemplo:**

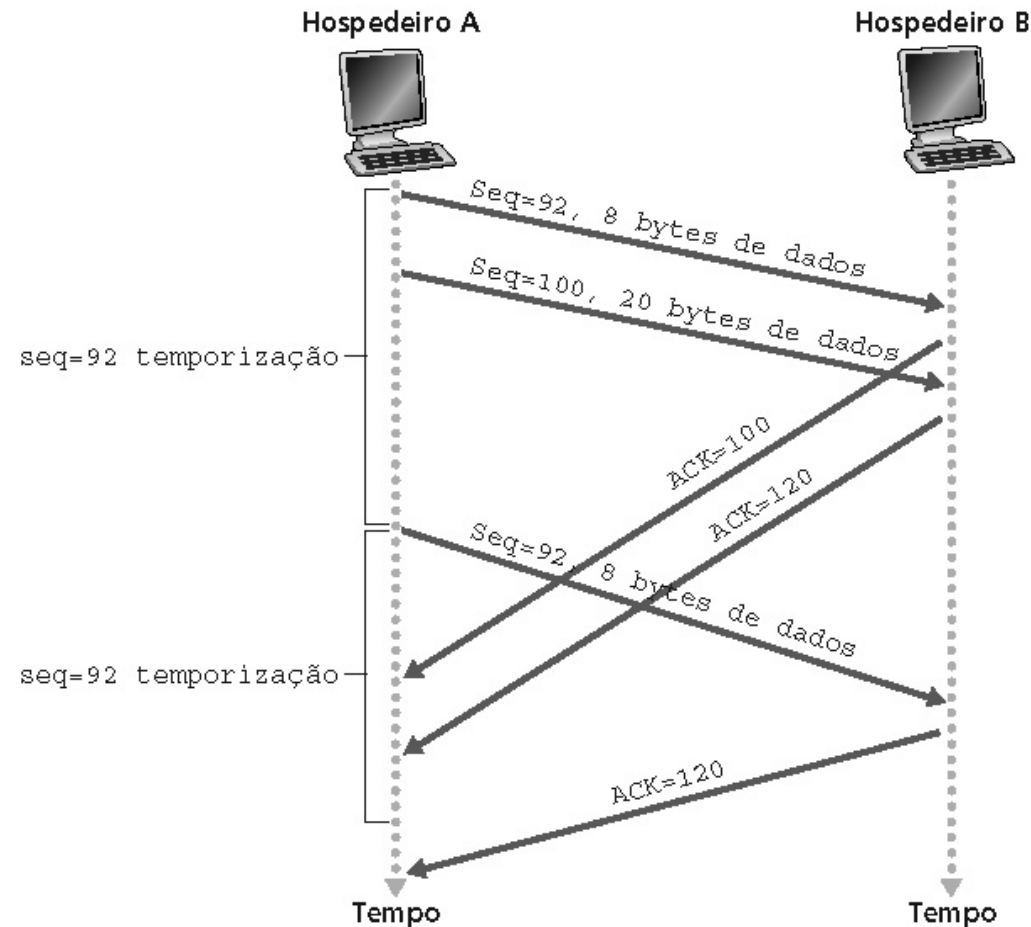
- SendBase-1 =  
71; y= 73,  
então o  
receptor  
deseja  
73+ ; y >  
SendBase,  
então  
o novo dado é  
confirmado



# 3 TCP: cenários de retransmissão



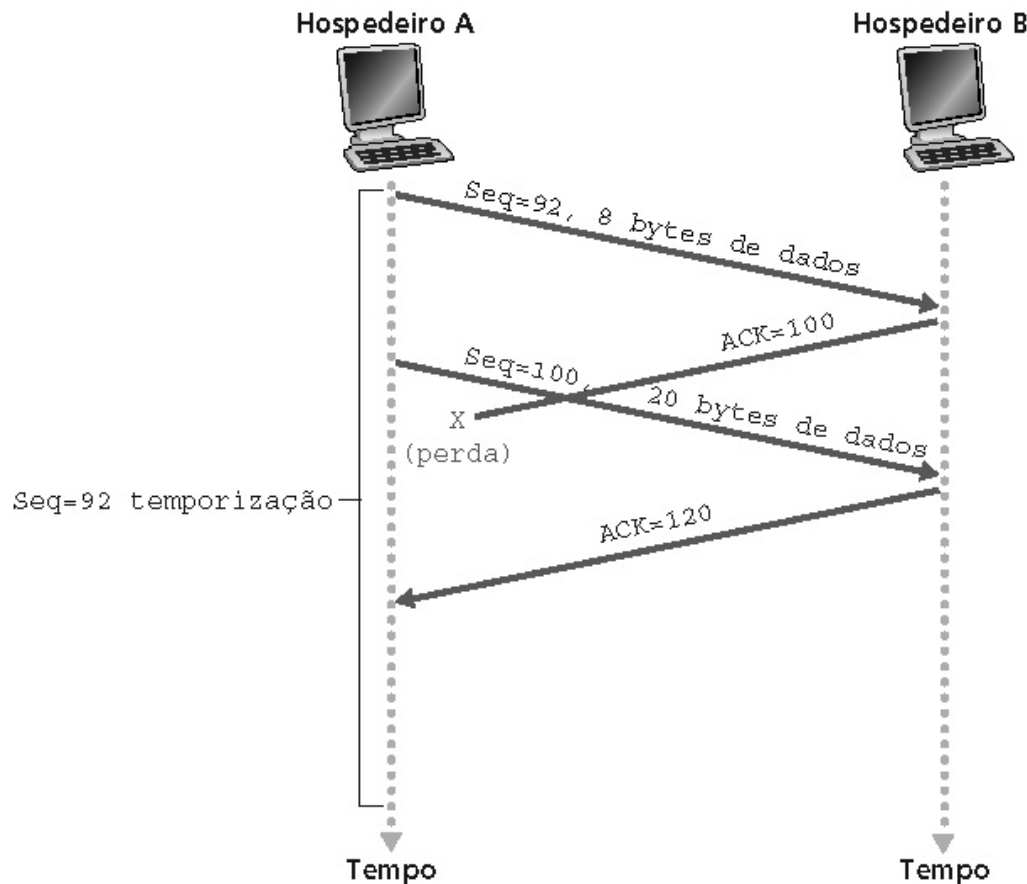
Cenário com perda do ACK



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 frequência, o tempo de expiração é relativamente longo:
  - Longo atraso antes de reenviar um pacote perdido
- Detecta segmentos perdidos por meio de ACKs duplicados
  - Transmissor frequentemente 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

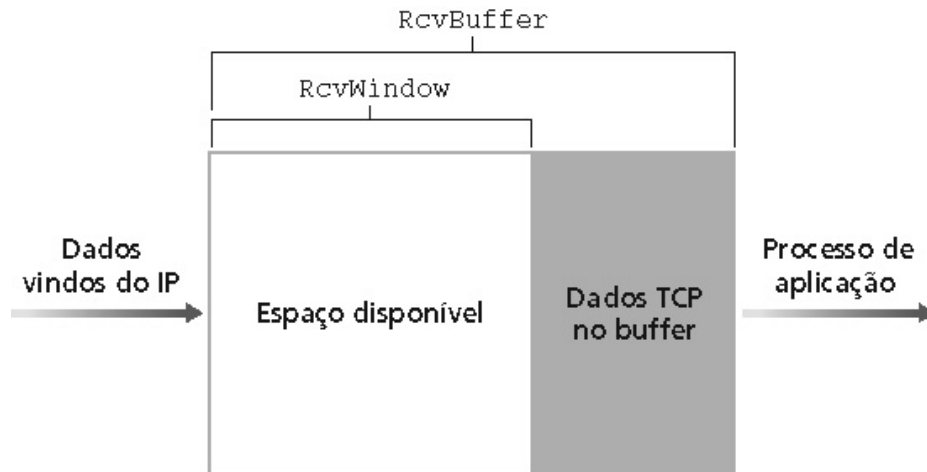
retransmissão rápida

# 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:



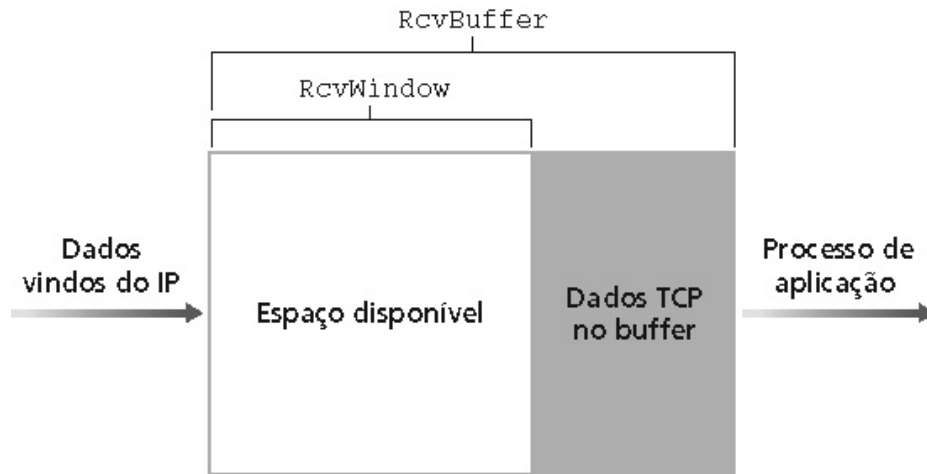
- Processos de aplicação podem ser lentos para ler o buffer

## 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 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 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:** sistema final cliente reconhece o SYNACK



PEARSON

Addison  
Wesley

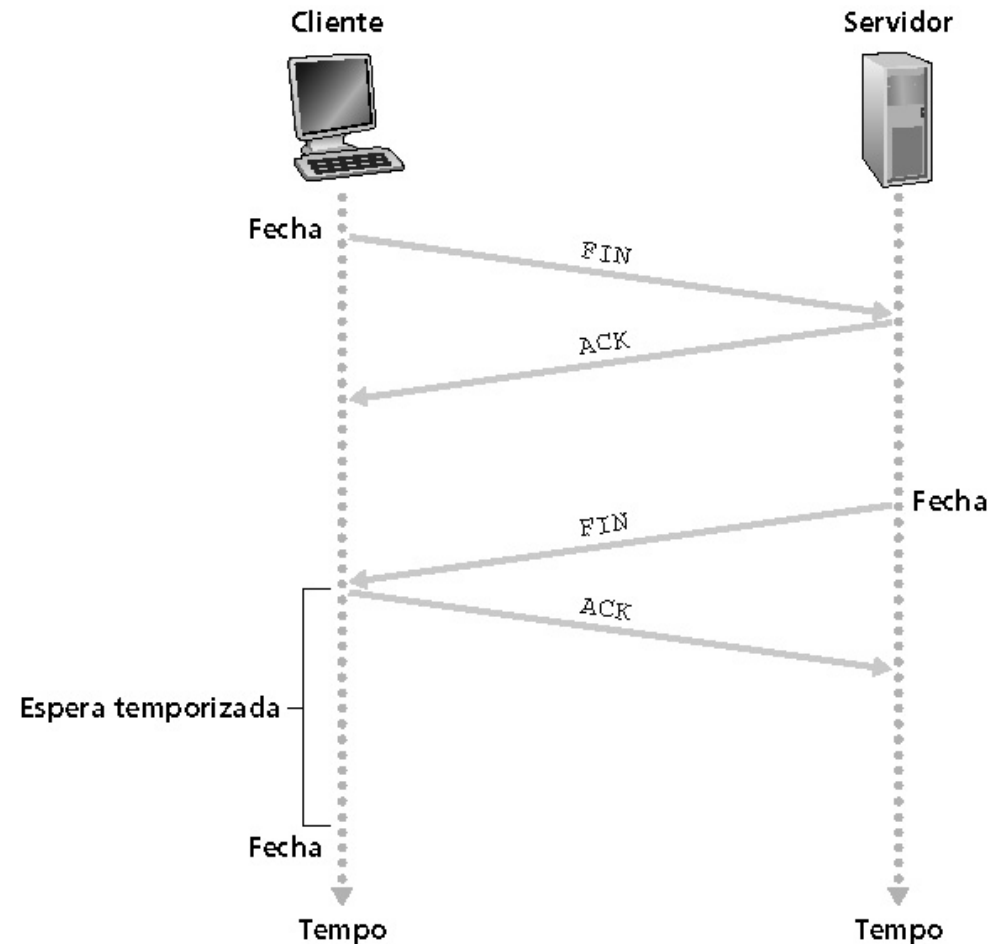
# 3 Gerenciamento de conexão TCP

## 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



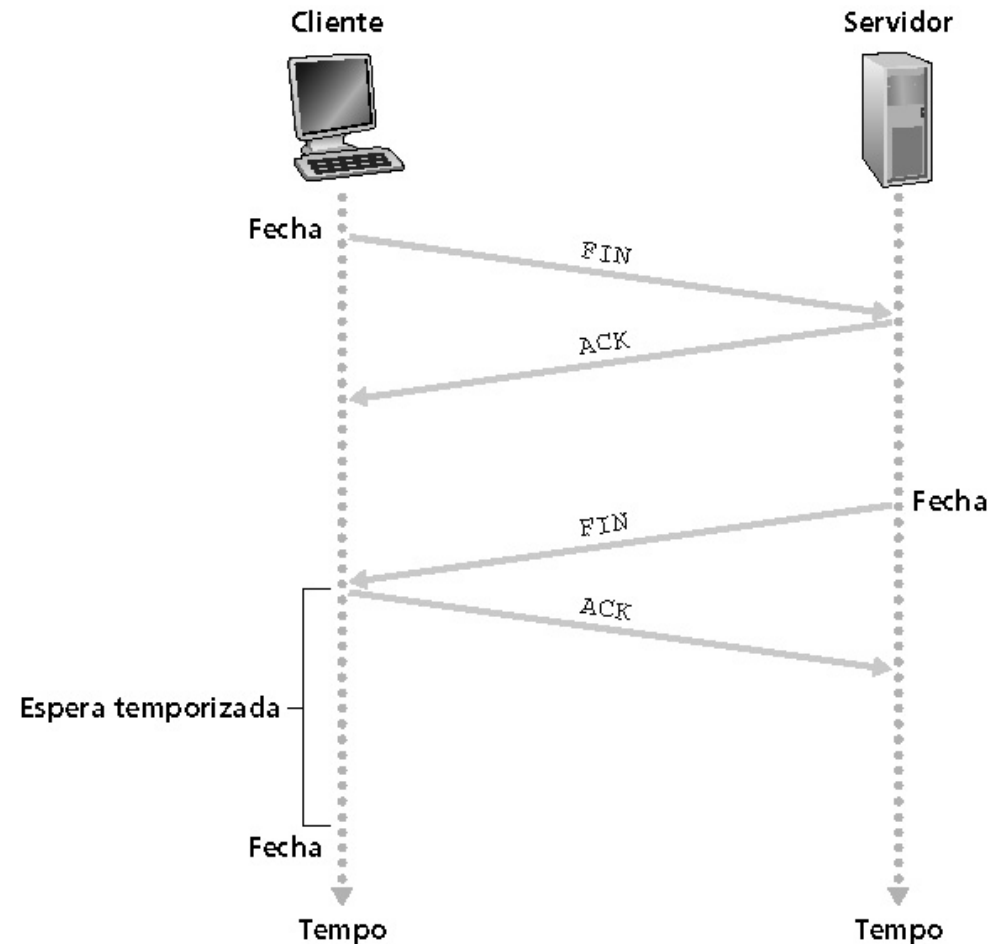
# 3 Gerenciamento de conexão TCP

**Passo 3:** cliente recebe FIN, responde com ACK

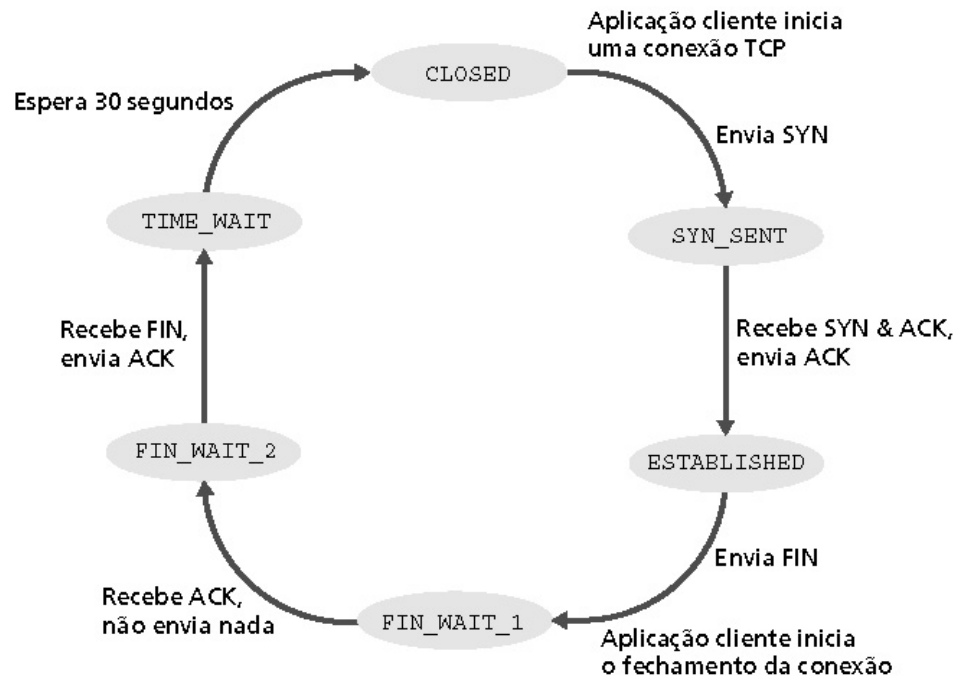
- Entra “espera temporizada” - vai responder com ACK a FINs recebidos

**Passo 4:** servidor, recebe ACK  
Conexão fechada

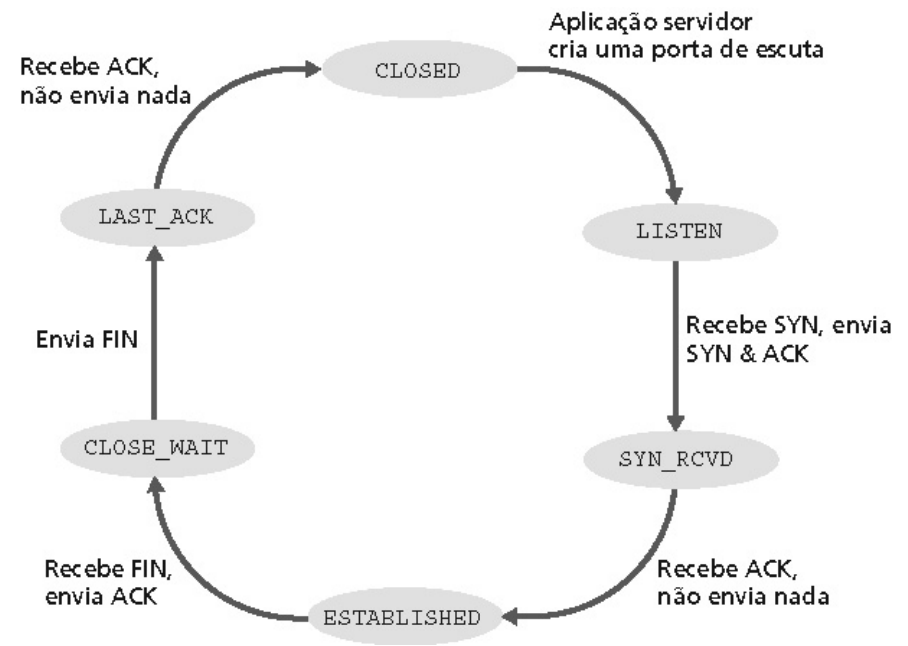
**Nota:** com uma pequena modificação, pode-se manipular FINs simultâneos



# 3 Gerenciamento de conexão TCP



Estados do cliente



Estados do servidor

# 3 Camada de transporte

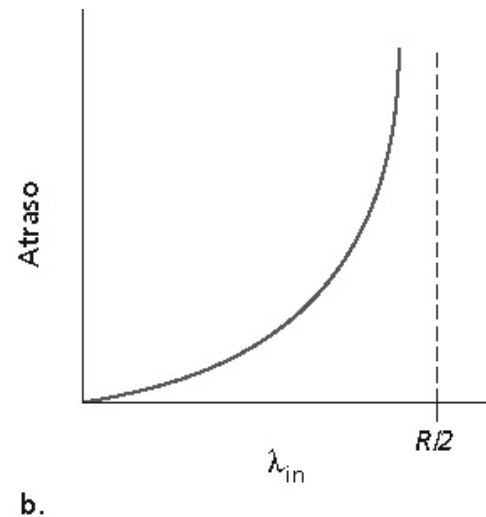
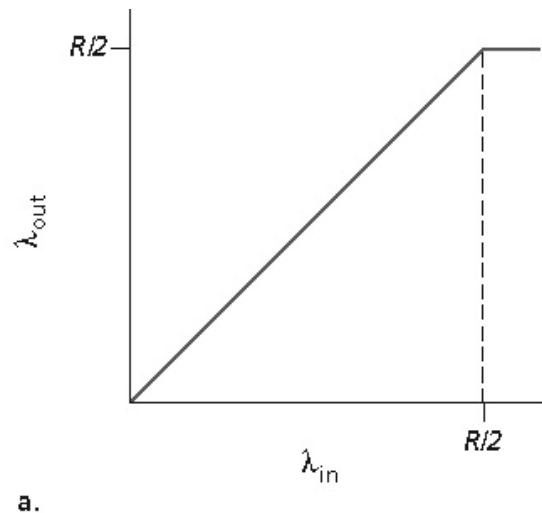
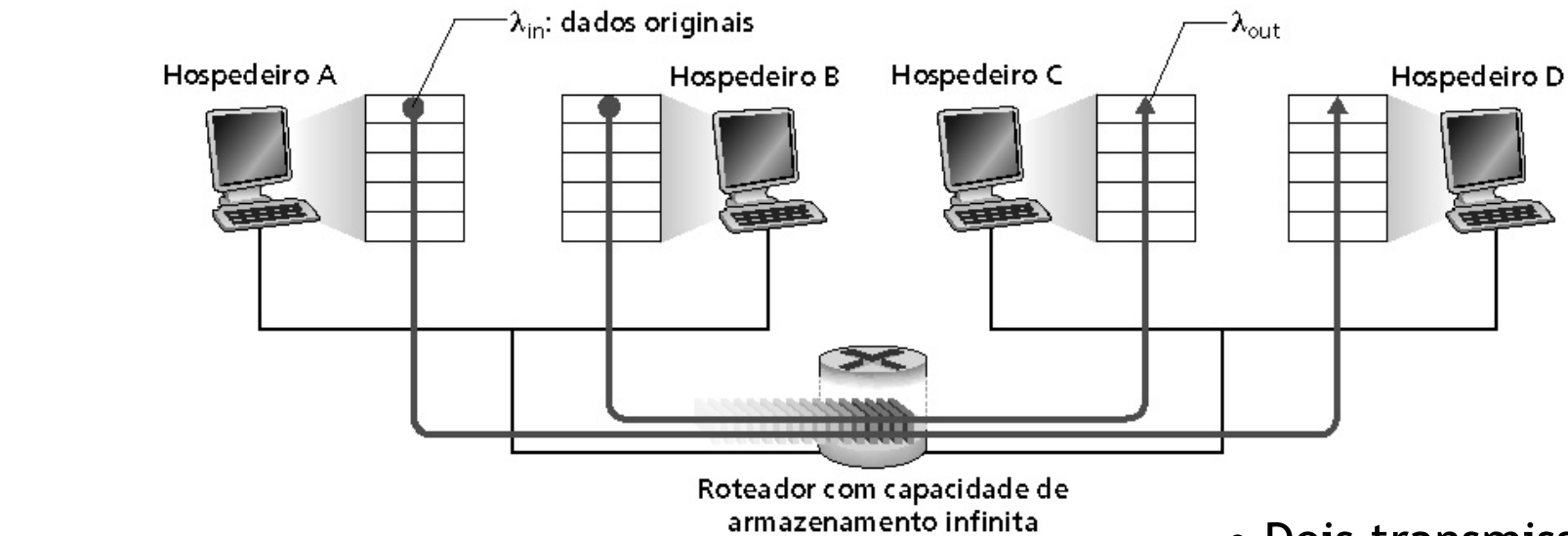
- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado à conexão: UDP
- 3.4 Princípios de transferência confiável de dados
- 3.5 Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo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

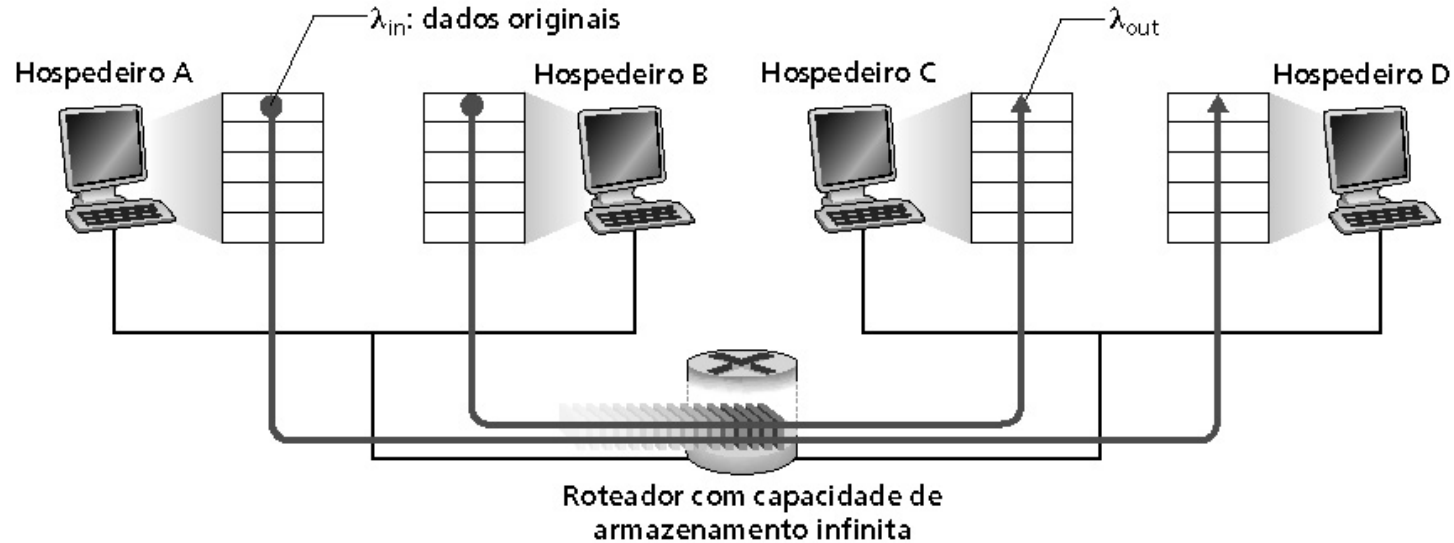
# 3 Causas / custos do congestionamento: cenário 1



- Dois transmissores, dois receptores
- Um roteador, buffers infinitos
- Não há retransmissão
- Grandes atrasos quando congestionado
- Máxima vazão alcançável

# 3 Causas / custos do congestionamento: cenário 2

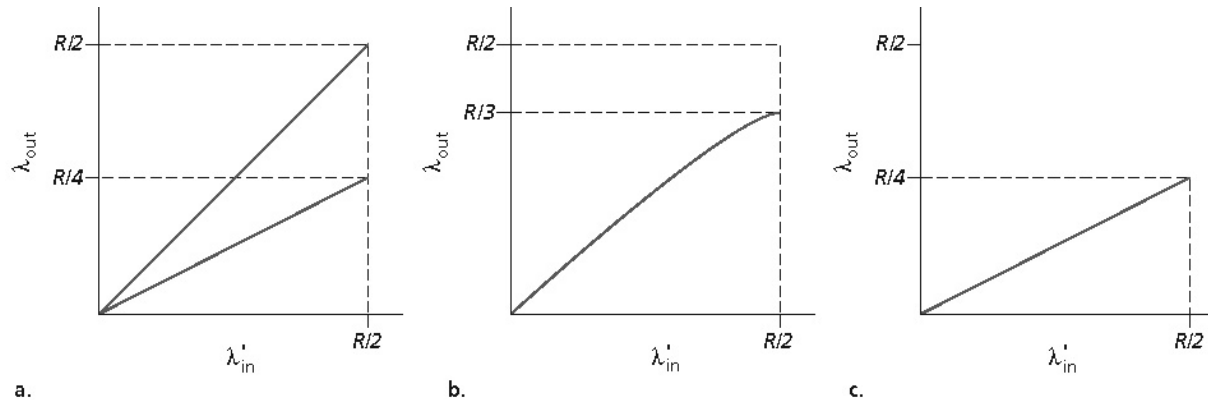
- Um roteador, buffers **finitos**
- Transmissor reenvia pacotes perdidos





# 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  $\lambda'_{in}$  maior (que o caso perfeito ) para o mesmo  $\lambda_{out}$



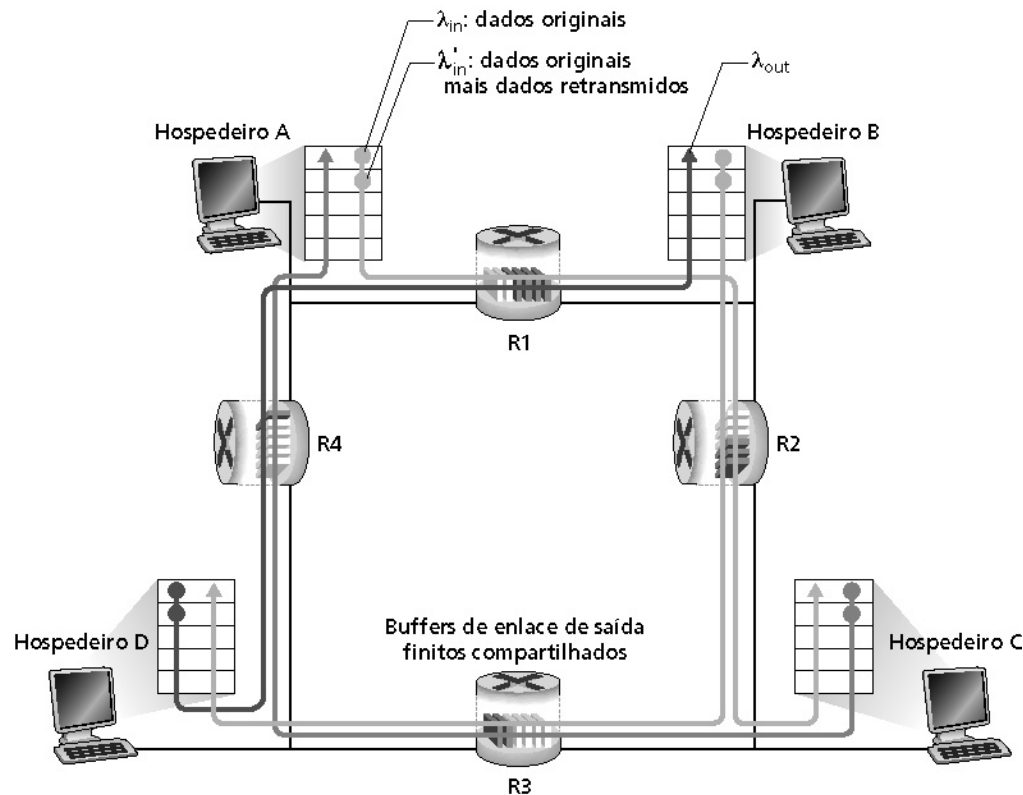
## “custos” do congestionamento:

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

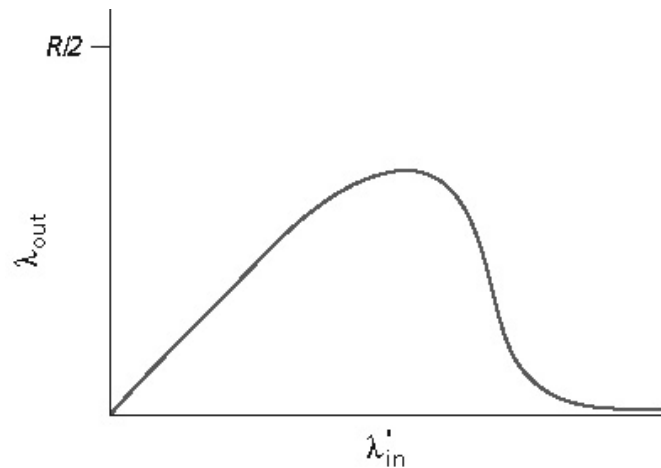
# 3 Causas / custos do congestionamento: cenário 3

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

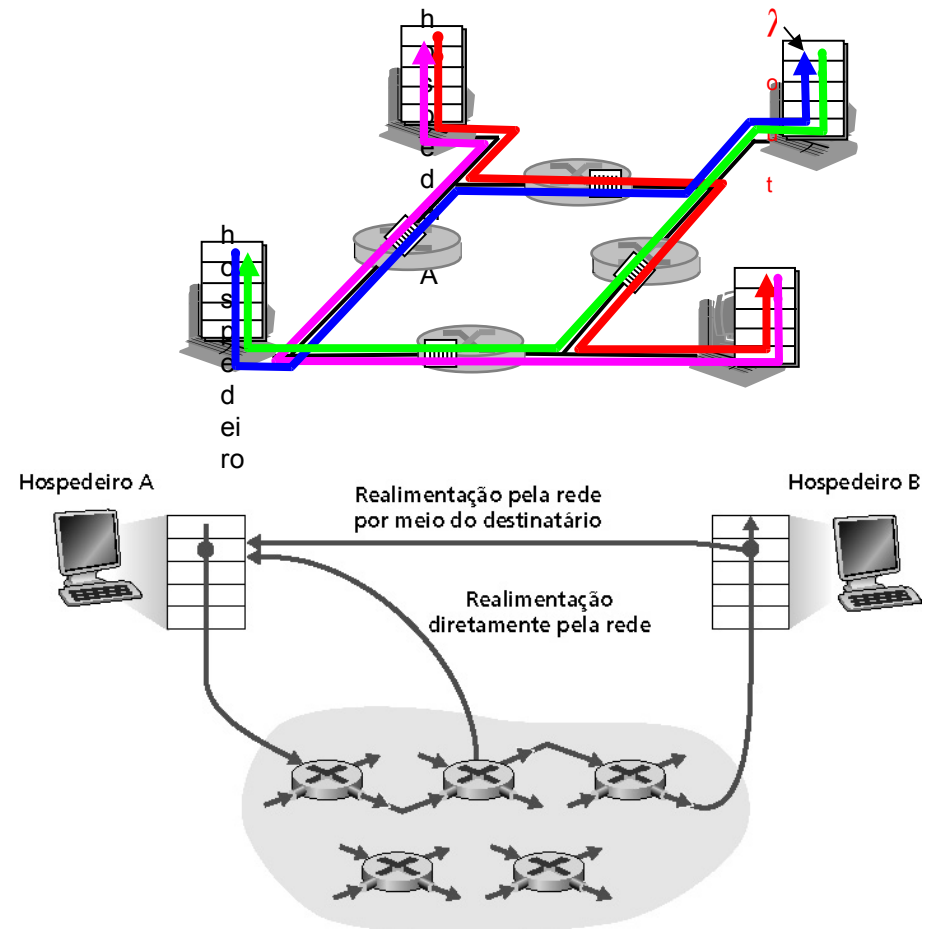
P.: O que acontece quando  $\lambda_{in}$  e  $\lambda'_{in}$  aumentam?



# 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

## ABR: available bit rate:

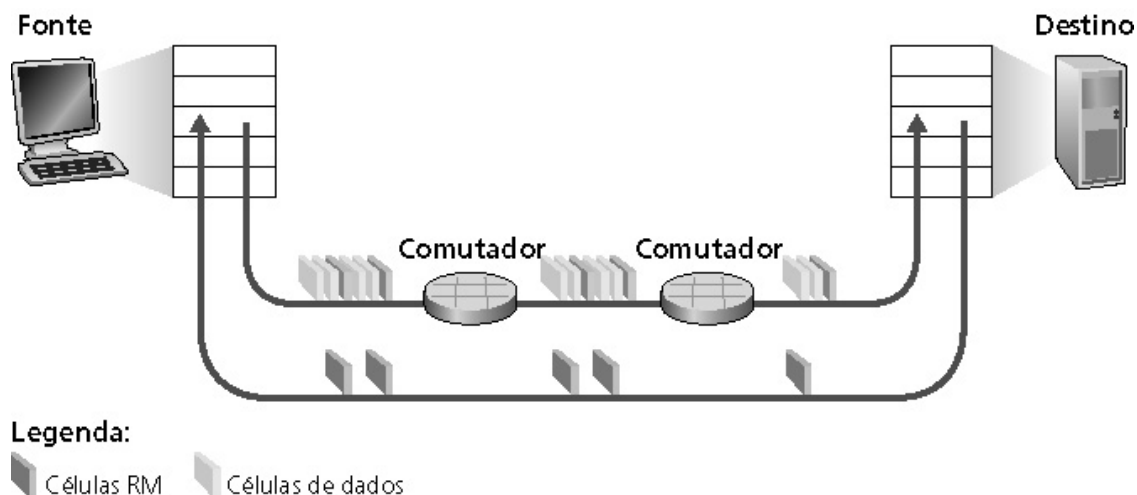
- “serviço elástico”
- Se o caminho do transmissor está pouco usado:
  - Transmissor pode usar a banda disponível
- Se o caminho do transmissor está congestionado:
  - Transmissor é limitado a uma taxa mínima garantida

## 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:  
$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$$

- Aproximadamente,

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

- **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:

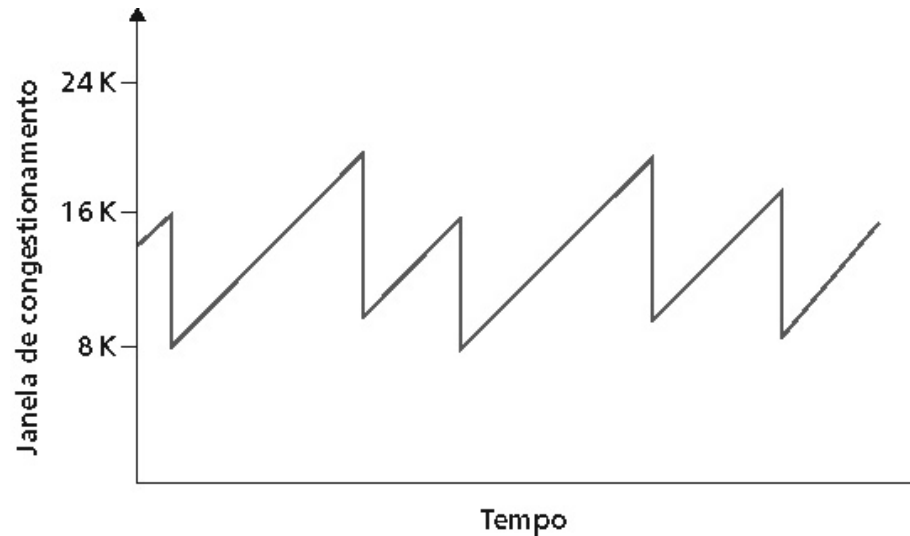
- AIMD
- Partida lenta
- Reação a eventos de esgotamento de temporização



# 3 TCP AIMD

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

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



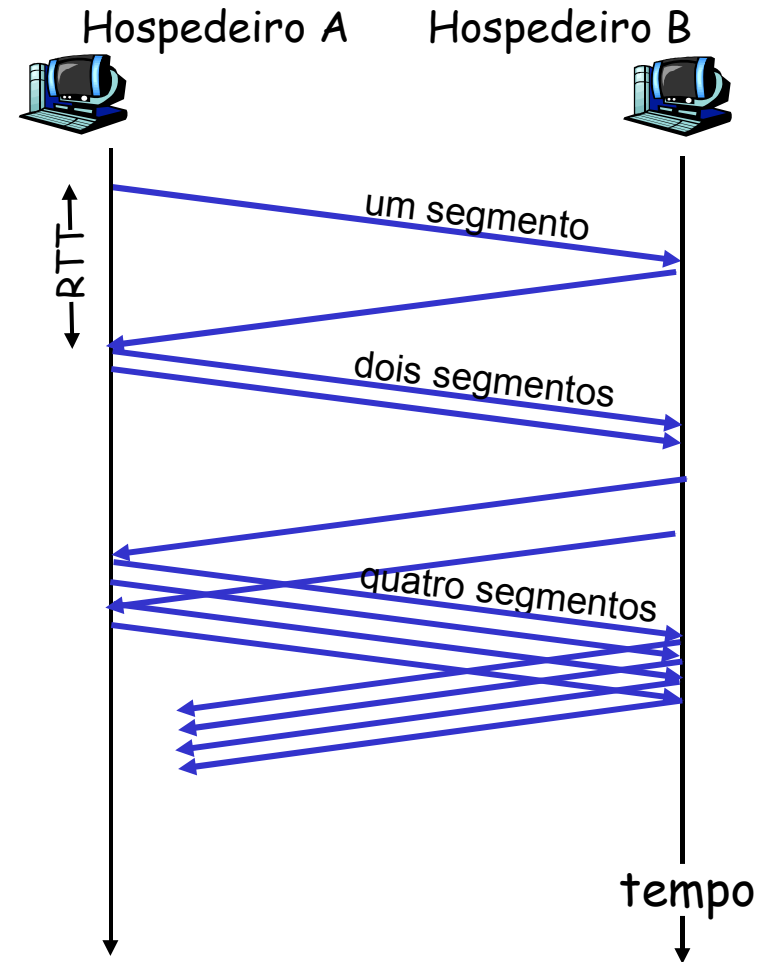
conexão TCP de longa-vida

# 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  $\gg$  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”

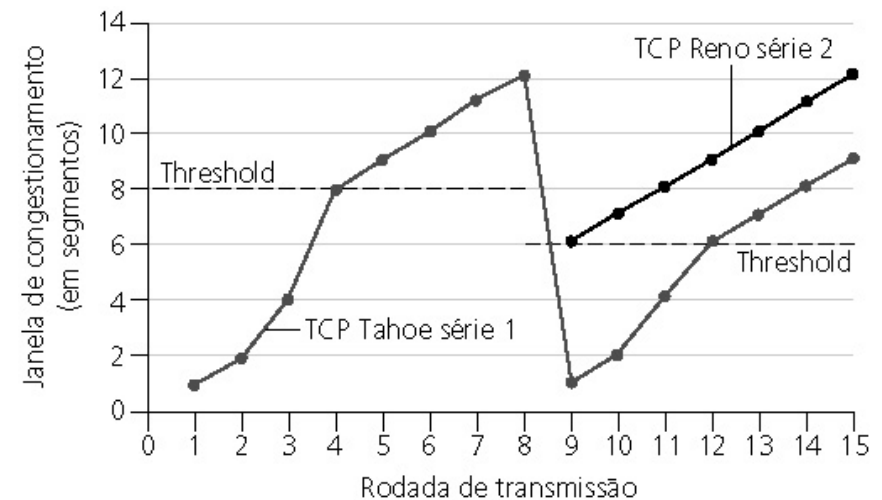
# 3 Refinamento

**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)	$\text{CongWin} = \text{CongWin} + \text{MSS}$ , If ( $\text{CongWin} > \text{Threshold}$ ) ajusta estado para “prevenção de congestionamento”	Resulta em dobrar o CongWin a cada RTT
ACK recebido para dado previamente não confirmado	prevenção de congestionamento (CA)	$\text{CongWin} = \text{CongWin} + \text{MSS} * (\text{MSS} / \text{CongWin})$	Aumento aditivo, resulta no aumento do CongWin em 1 MSS a cada RTT
Evento de perda detectado por três ACKs duplicados	SS or CA	$\text{Threshold} = \text{CongWin} / 2$ , $\text{CongWin} = \text{Threshold}$ , ajusta estado para “prevenção de congestionamento”	Recuperação rápida, implementando redução multiplicativa o CongWin não cairá abaixo de 1 MSS.
Tempo de confirmação	SS or CA	$\text{Threshold} = \text{CongWin} / 2$ , $\text{CongWin} = 1 \text{ MSS}$ , 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 não mudam

# 3 TCP throughput

- O que é **throughput** 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  $W/\text{RTT}$
- Logo após a perda, a janela cai para  $W/2$ , e o **throughput** para  $W/2\text{RTT}$
- **Throughput** médio:  $0,75 W/\text{RTT}$





# 3 Futuro do TCP

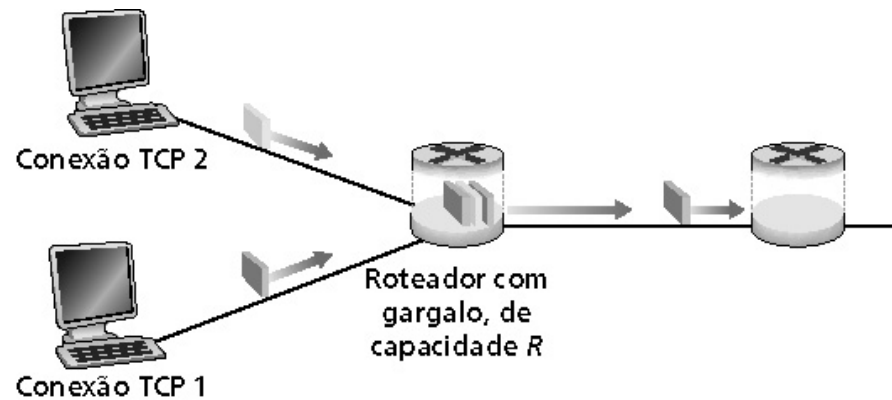
- Exemplo: segmento de 1500 bytes, RTT de 100 ms, deseja 10 Gbps de *throughput*
- Requer tamanho de janela  $W = 83,333$  para os segmentos em trânsito
- Throughput em termos da taxa de perda:

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

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

# 3 Equidade do TCP

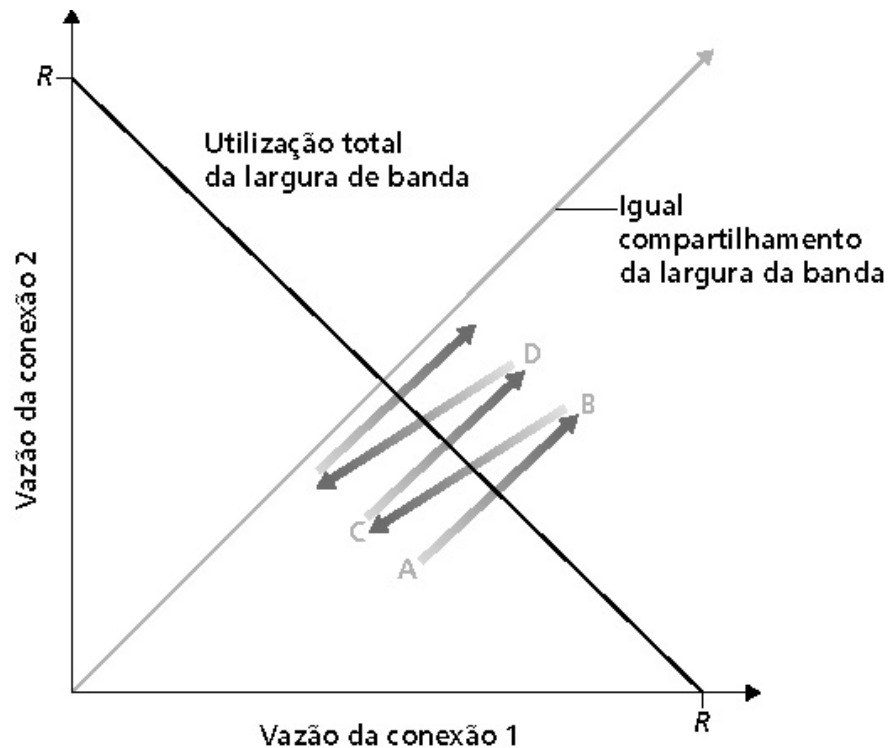
**Objetivo de equidade:** se  $K$  sessões TCP compartilham o mesmo enlace do gargalo com largura de banda  $R$ , cada uma deve ter taxa média de  $R/K$



# 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 2  
prevenção de congestionamento:  
aumento aditivo  
perda: reduz janela por um fator de 2  
prevenção de congestionamento:  
aumento aditivo

# 3 Equidade

## Equidade 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

## Equidade 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  $R/2$ !

# 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)

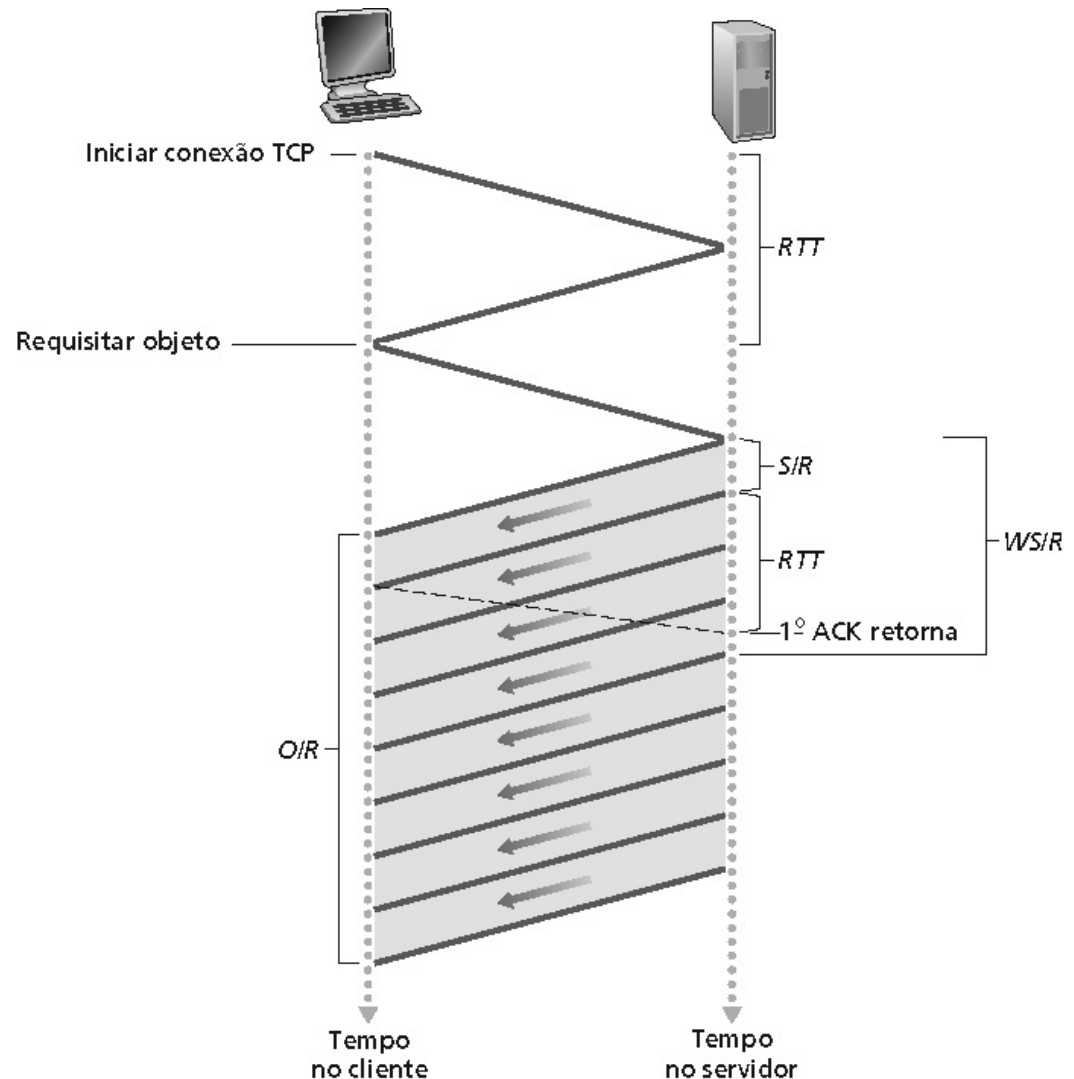
**Tamanho da janela:**

- 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



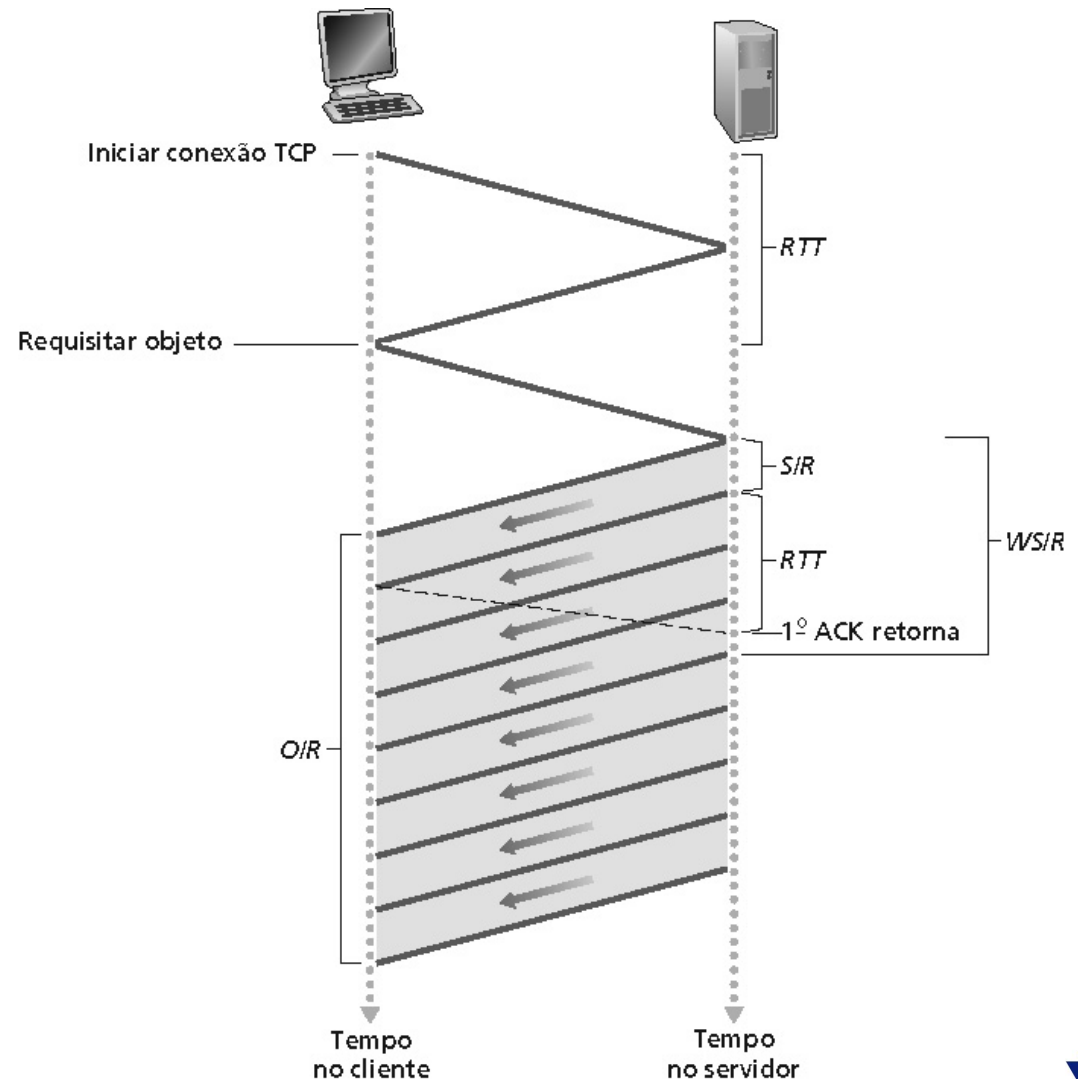
$$\text{atraso} = 2RTT + O/R$$

# 3 Janela de congestionamento fixa (2)

## Segundo caso:

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

$$\text{atraso} = 2RTT + O/R + (K-1)[S/R + RTT - WS/R]$$



# 3 TCP Modelagem de latência: partida lenta (1)

- Agora suponha que a janela cresça 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**

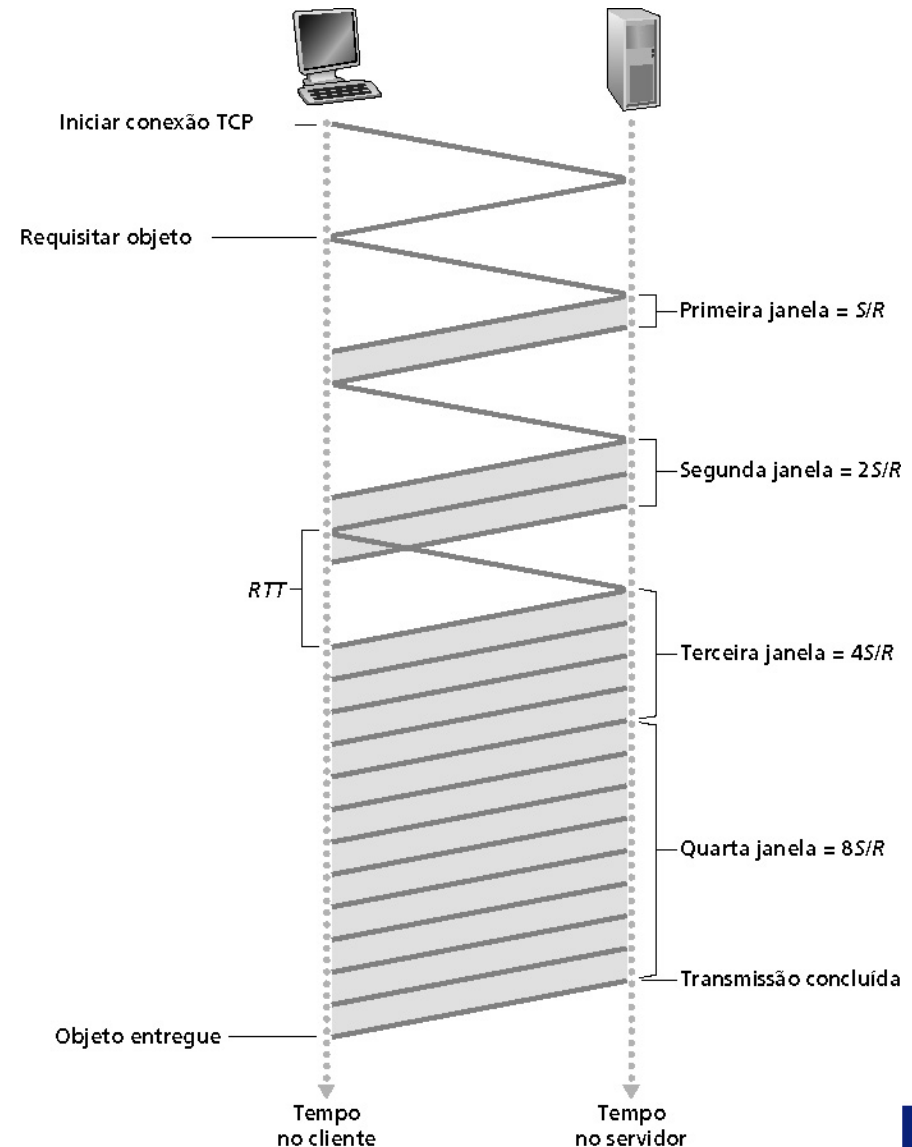
## Servidor inativo:

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

## Exemplo:

- O/S = 15 segmentos
  - $K = 4$  janelas
  - $Q = 2$
- $v P = \min\{K-1, Q\} = 2$

Servidor inativo  $P = 2$  tempos



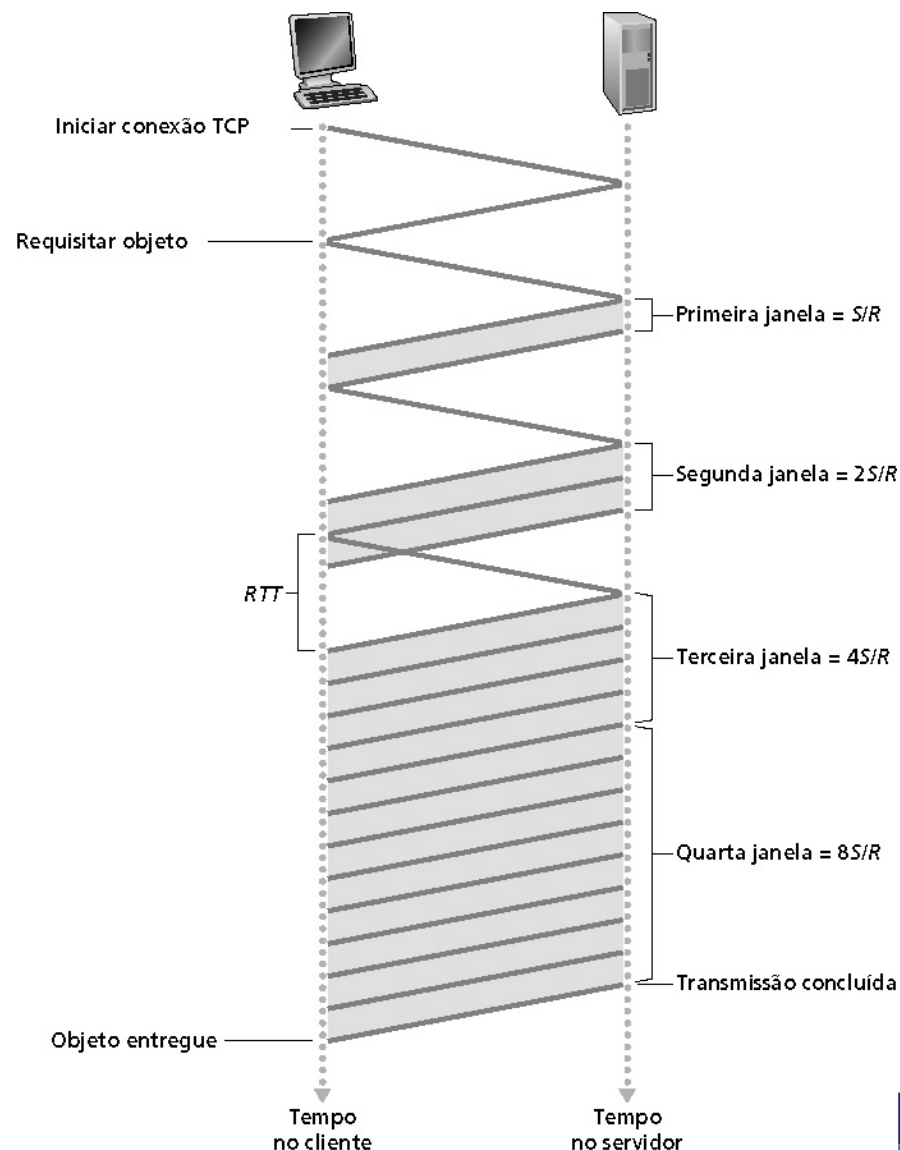
# 3 TCP modelagem de latência: partida lenta (3)

$\frac{S}{R} + RTT =$  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

$$\begin{aligned} \text{latência} &= \frac{O}{R} + 2RTT + \sum_{p=1}^P \text{TempoBloqueio}_p \\ &= \frac{O}{R} + 2RTT + \sum_{k=1}^P \left[ \frac{S}{R} + RTT - 2^{k-1} \frac{S}{R} \right] \\ &= \frac{O}{R} + 2RTT + P \left[ RTT + \frac{S}{R} \right] - (2^P - 1) \frac{S}{R} \end{aligned}$$



# 3 TCP modelagem de latência: partida lenta (4)

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

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

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

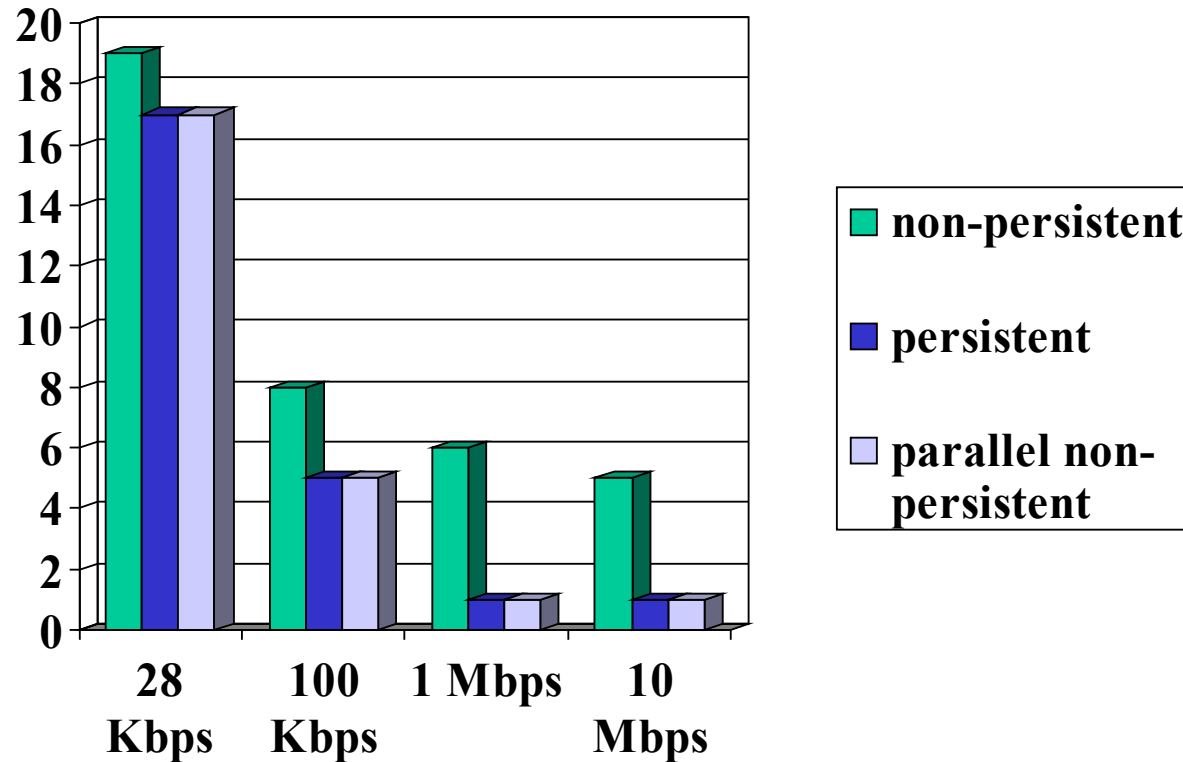
# 3 Modelagem HTTP

- 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

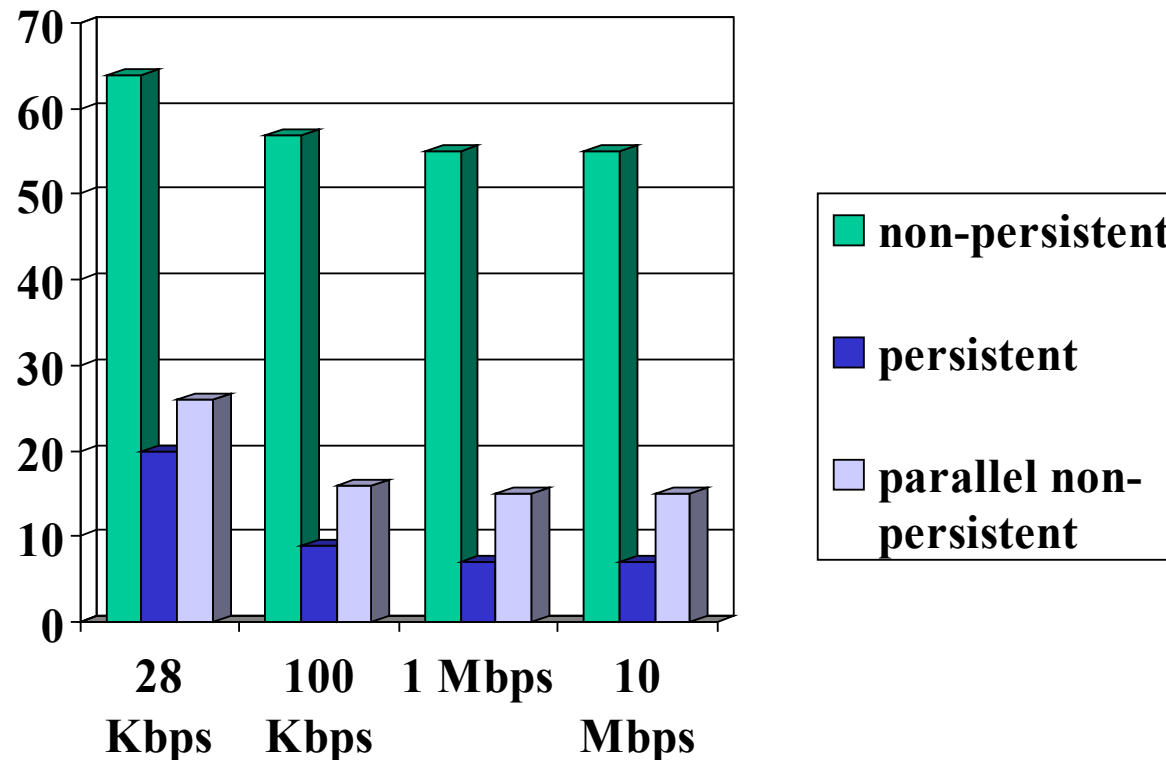


Para pouca largura de banda, tempo de conexão e resposta dominados pelo tempo de transmissão

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