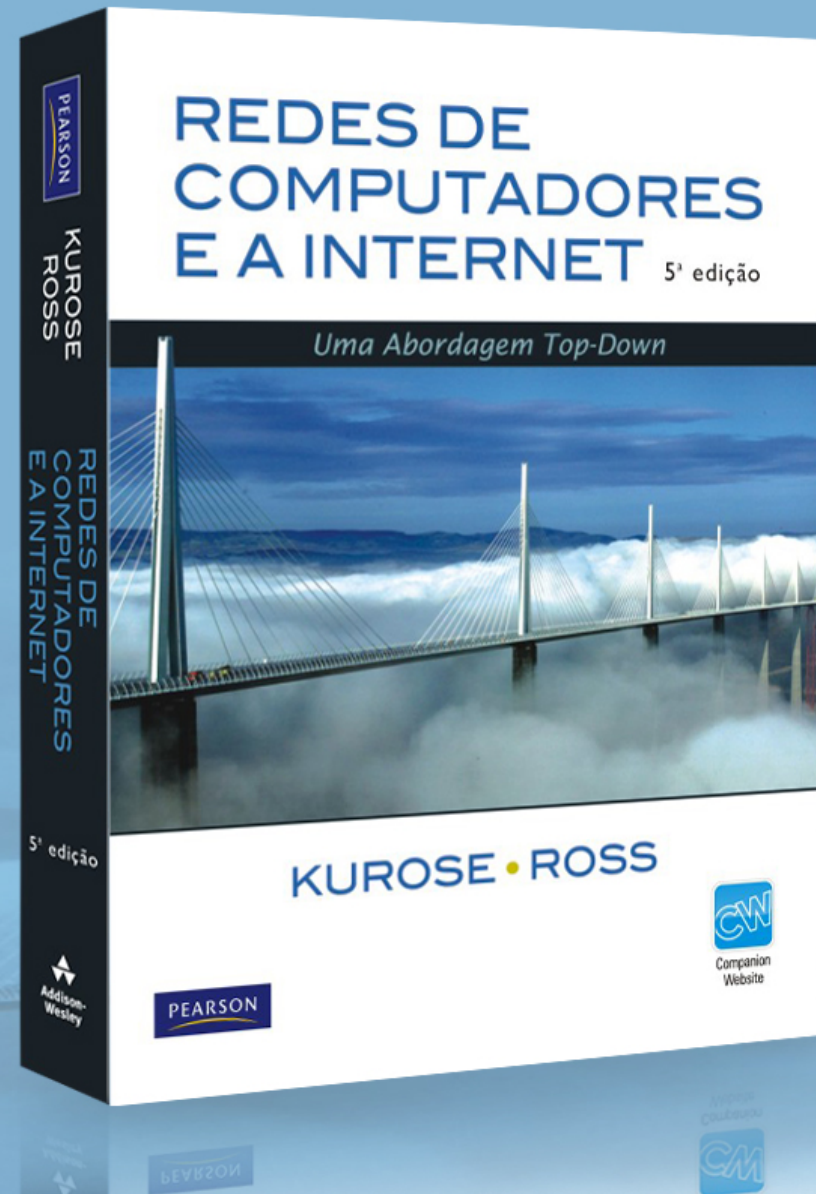


Capítulo 3 Camada de transporte



Capítulo 3: Camada de transporte

Objetivos do capítulo:

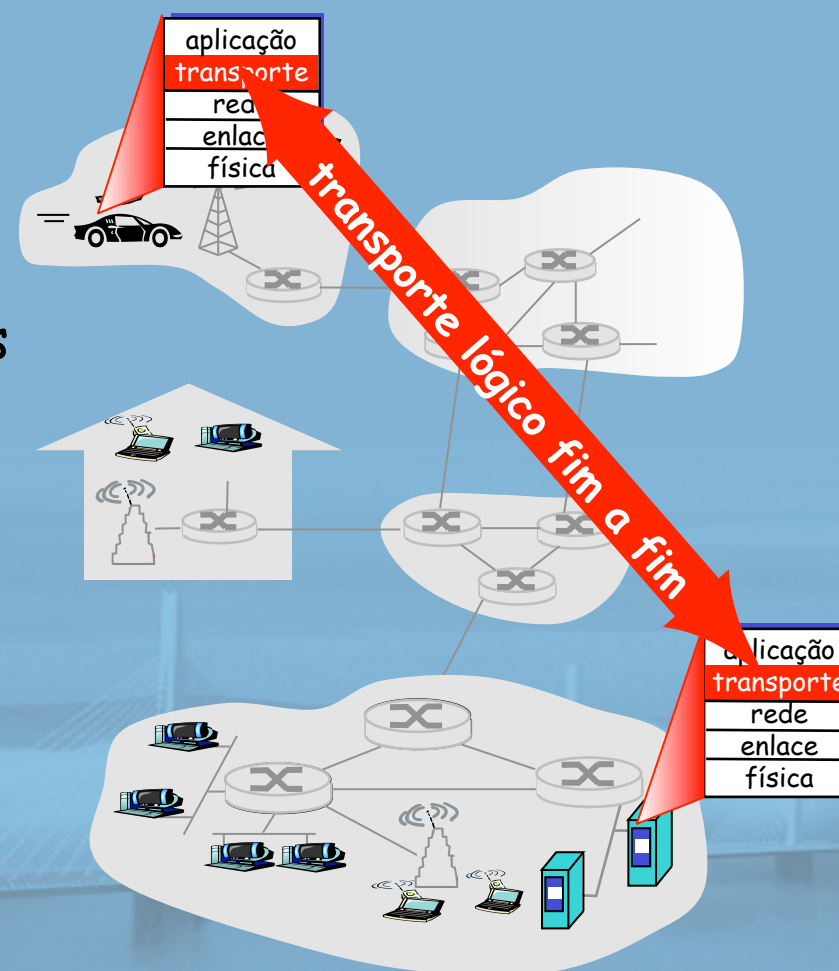
- ❑ princípios dos serviços da camada de transporte:
 - multiplexação/demultiplexação
 - transferência de dados confiável
 - controle de fluxo
 - controle de congestionamento
- ❑ protocolos da camada de transporte na Internet:
 - UDP: transporte sem conexão
 - TCP: transporte orientado a conexão
 - controle 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

Serviços e protocolos de transporte

- ❑ oferecem **comunicação lógica** entre processos de aplicação rodando em hosts diferentes
- ❑ protocolos de transporte rodam em **sistemas finais**
 - lado remetente: divide as msgs da aplicação em **segmentos**, passa à camada de rede
 - lado destinatário: remonta os segmentos em msgs, passa à camada de aplicação
- ❑ mais de um protocolo de transporte disponível às aplicações
 - Internet: TCP e UDP



Camada de transporte versus rede

- ❑ *camada de rede:*
comunicação lógica
entre hosts
- ❑ *camada de transporte:*
comunicação lógica
entre processos
 - conta com e amplia os
serviços da camada de
rede

analogia com a família:

*12 crianças mandando
cartas a 12 crianças*

- ❑ processos = crianças
- ❑ msgs da aplicação =
cartas nos envelopes
- ❑ hosts = casas
- ❑ protocolo de transporte
= Ana e Bill
- ❑ protocolo da camada de
rede = serviço postal

Camada de transporte versus rede

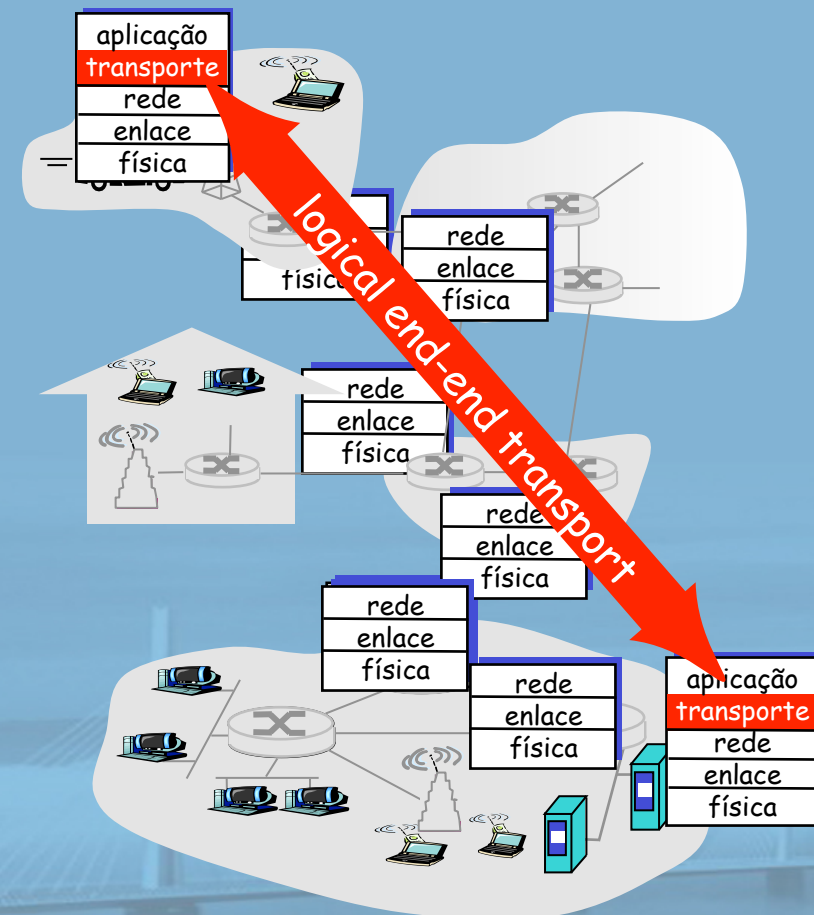
- ❑ *camada de rede:*
comunicação lógica entre _____
- ❑ *camada de transporte:*
comunicação lógica entre _____

Protocolos da camada de transporte da Internet

REDES DE
COMPUTADORES
E A INTERNET 5ª edição

Uma Abordagem Top-Down

- ❑ remessa confiável e em ordem (TCP)
 - controle de congestionamento
 - controle de fluxo
 - estabelecimento da conexão
- ❑ remessa não confiável e desordenada: UDP
 - extensão sem luxo do IP pelo "melhor esforço"
- ❑ serviços não disponíveis:
 - garantias de atraso
 - garantias de largura de banda



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

Multiplexação/ demultiplexação

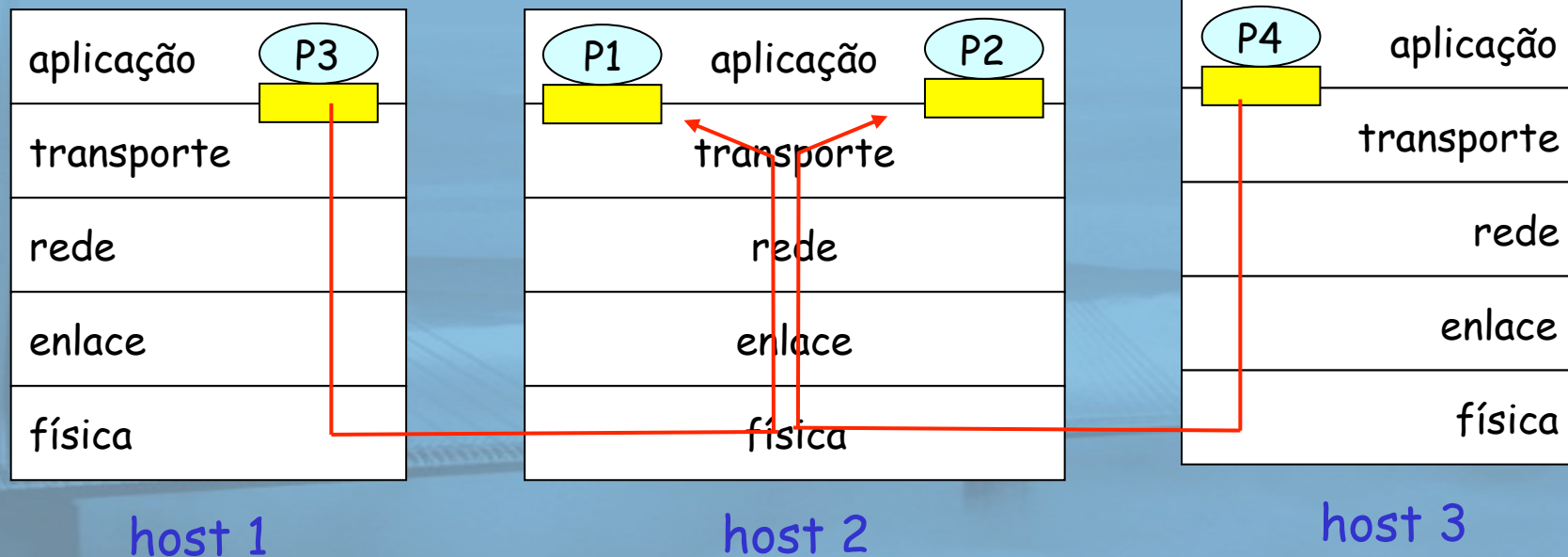
demultiplexação no destinatário:

entregando segmentos
recebidos ao socket correto

 = socket  = processo

multiplexação no remetente:

colhendo dados de múltiplos
sockets, envelopando dados
com cabeçalho (usados depois
para demultiplexação)

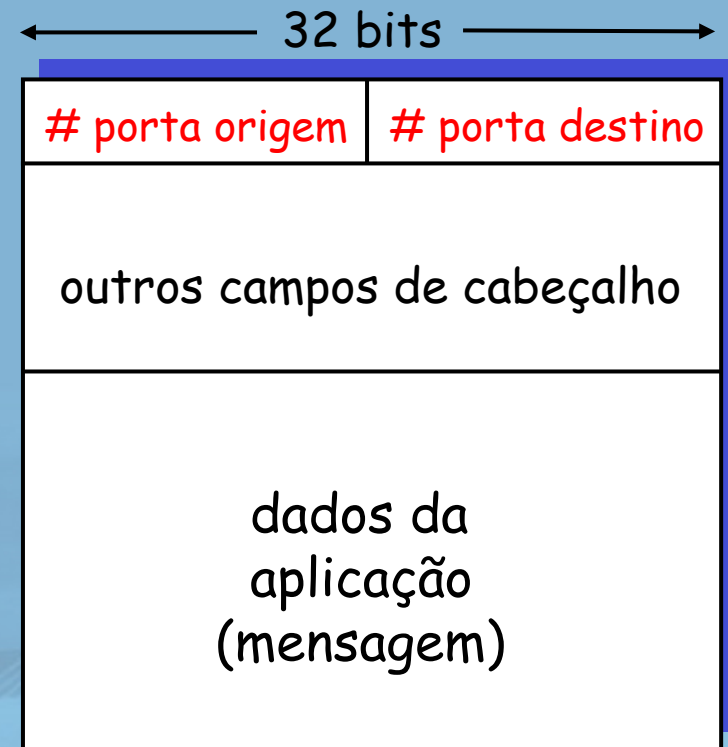


Como funciona a demultiplexação

□ host recebe datagramas IP

- cada datagrama tem endereço IP de origem, endereço IP de destino
- cada datagrama carrega 1 segmento da camada de transporte
- cada segmento tem número de porta de origem, destino

□ host usa endereços IP & números de porta para direcionar segmento ao socket apropriado



formato do segmento TCP/UDP

Demultiplexação não orientada para conexão

- ❑ cria sockets com números de porta:

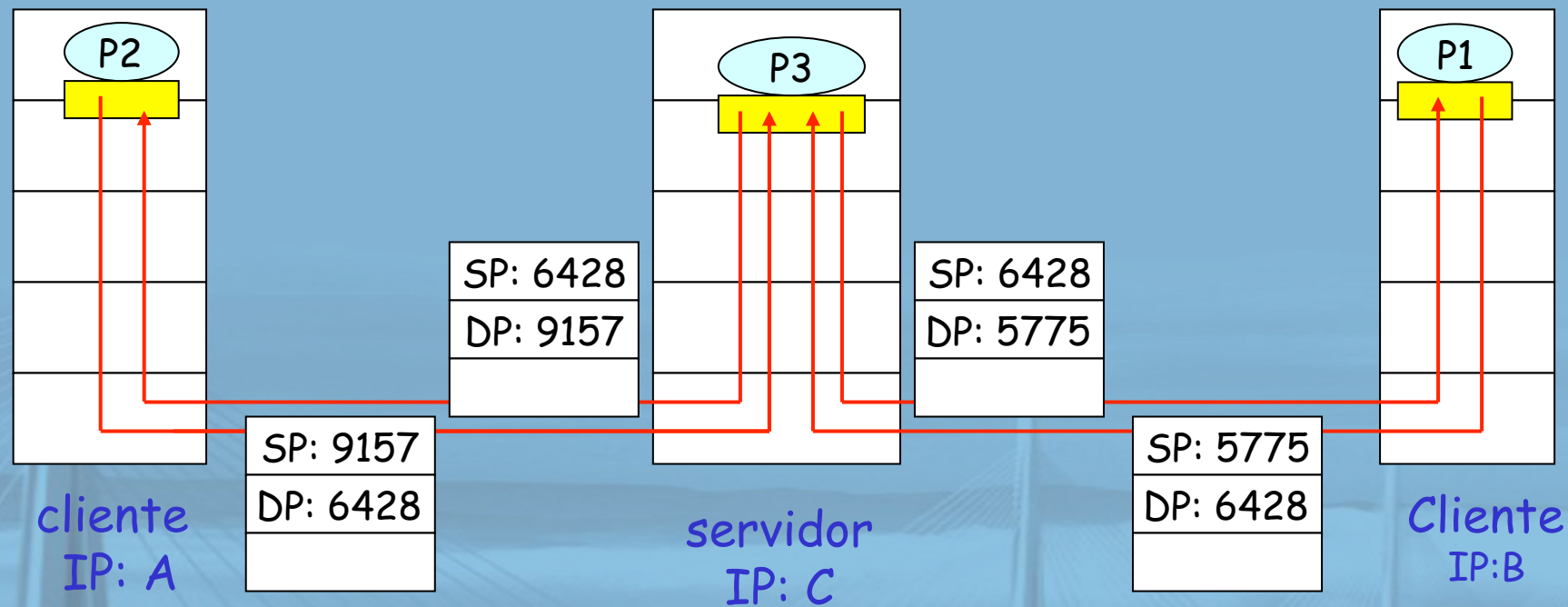
```
DatagramSocket mySocket1 = new  
    DatagramSocket(12534);
```

```
DatagramSocket mySocket2 = new  
    DatagramSocket(12535);
```

- ❑ socket UDP identificado por tupla de dois elementos:
(endereço IP destino, número porta destino)

- ❑ quando o host recebe segmento UDP:
 - verifica número de porta de destino no segmento
 - direciona segmento UDP para socket com esse número de porta
- ❑ datagramas IP com diferentes endereços IP de origem e/ou números de porta de origem são direcionados para o mesmo socket

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



SP oferece "endereço de retorno"

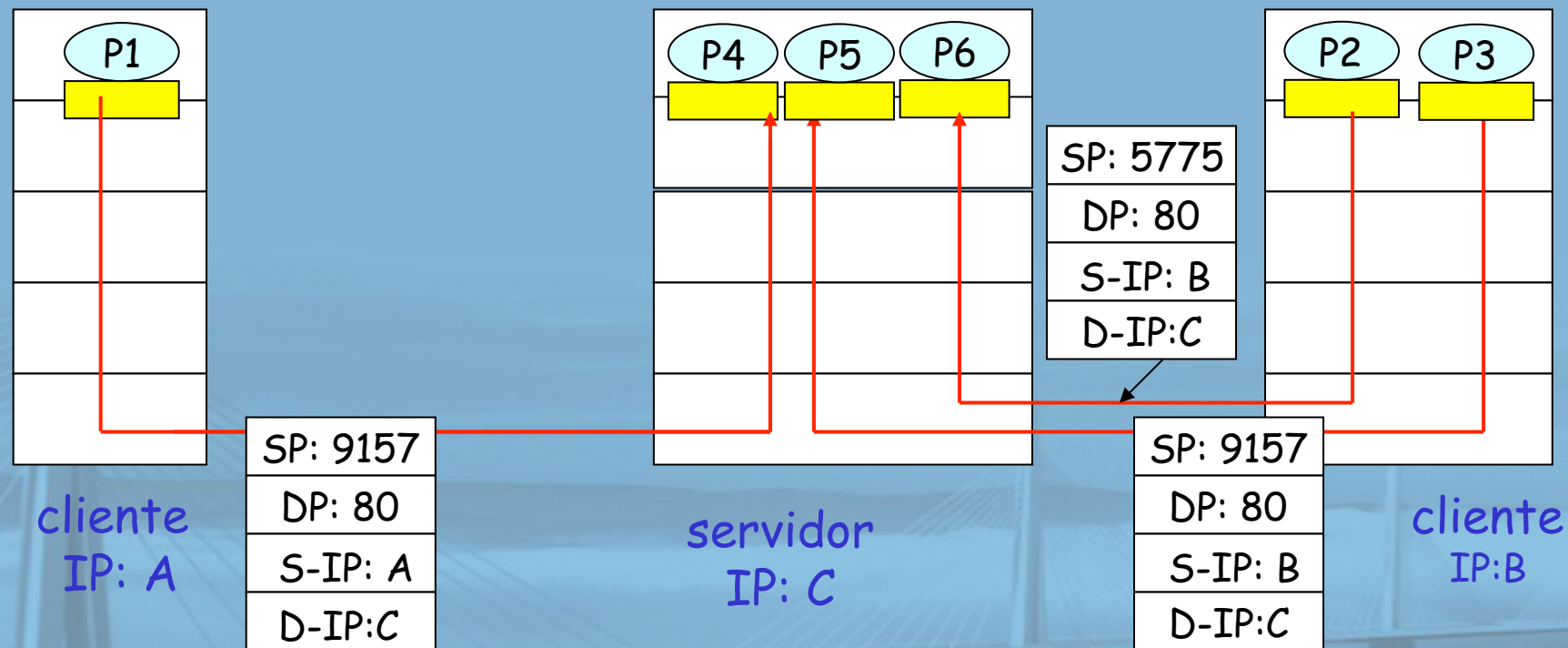
Demultiplexação orientada para conexão

- ❑ socket TCP identificado por tupla de 4 elementos:
 - endereço IP de origem
 - número de porta de origem
 - endereço IP de destino
 - número de porta de destino
- ❑ host destinatário usa todos os quatro valores para direcionar segmento ao socket apropriado
- ❑ host servidor pode admitir muitos sockets TCP simultâneos:
 - cada socket identificado usa a própria tupla de 4 elementos
- ❑ servidores Web têm diferentes sockets para cada cliente conectando
 - HTTP não persistente terá diferentes sockets para cada requisição

REDES DE COMPUTADORES E A INTERNET

5ª edição

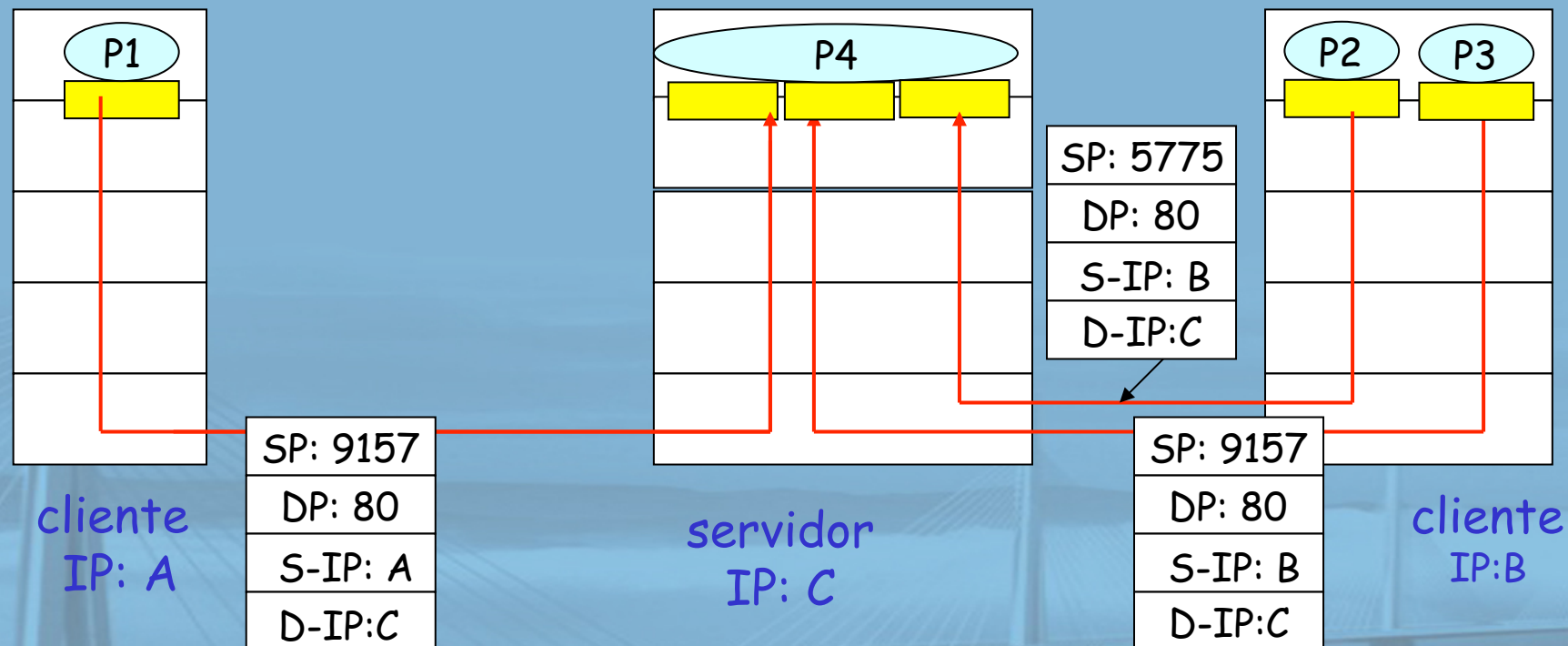
Uma Abordagem Top-Down



Demultiplexação orientada para conexão: servidor Web threaded

REDES DE
COMPUTADORES
E A INTERNET 5ª edição

Uma Abordagem Top-Down



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

UDP: User Datagram Protocol [RFC 768]

- ❑ protocolo de transporte da Internet "sem luxo", básico
- ❑ serviço de "melhor esforço", segmentos UDP podem ser:
 - perdidos
 - entregues à aplicação fora da ordem
- ❑ **sem conexão:**
 - sem handshaking entre remetente e destinatário UDP
 - cada segmento UDP tratado independente dos outros

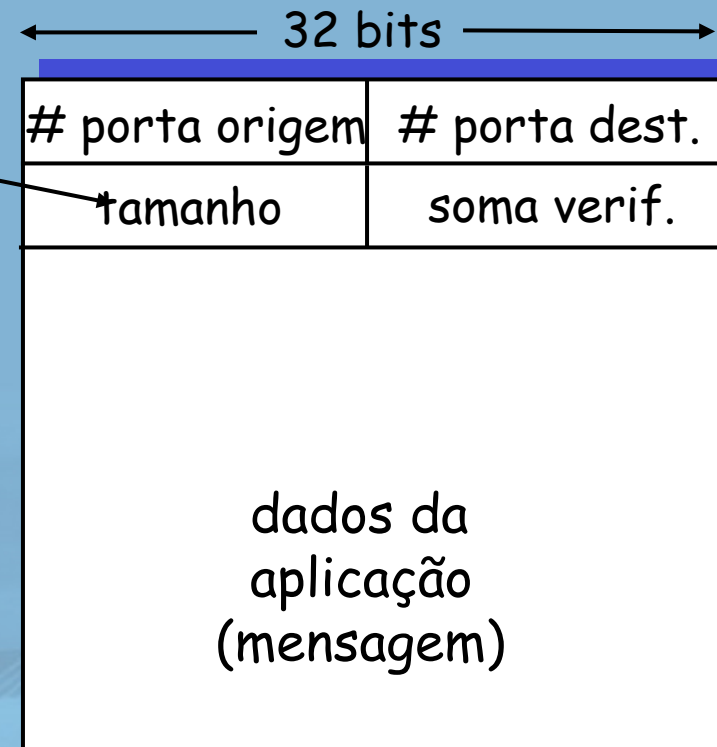
Por que existe um UDP?

- ❑ sem estabelecimento de conexão (que pode gerar atraso)
- ❑ simples: sem estado de conexão no remetente, destinatário
- ❑ cabeçalho de segmento pequeno
- ❑ sem controle de congestionamento: UDP pode transmitir o mais rápido possível

UDP: mais

- ❑ normalmente usado para streaming de aplicações de multimídia
 - tolerante a perdas
 - sensível à taxa
- ❑ outros usos do UDP
 - DNS
 - SNMP
- ❑ transferência confiável por UDP: aumenta confiabilidade na camada de aplicação
 - recuperação de erro específica da aplicação!

tamanho,
em bytes, do
segmento UDP,
incluindo
cabeçalho



formato de segmento UDP

Soma de verificação UDP

objetivo: detectar “erros” (p. e., bits invertidos) no segmento transmitido

remetente:

- ❑ trata conteúdo de segmento como sequência de inteiros de 16 bits
- ❑ soma de verificação (*checksum*): adição (soma por complemento de 1) do conteúdo do segmento
- ❑ remetente coloca valor da soma de verificação no campo de soma de verificação UDP

destinatário:

- ❑ calcula soma de verificação do segmento recebido
- ❑ verifica se soma de verificação calculada igual ao valor do campo de soma de verificação:
 - NÃO - erro detectado
 - SIM - nenhum erro detectado. *Mas pode haver erros mesmo assim? Veja mais adiante*

Exemplo de soma de verificação da Internet

□ nota

- Ao somar números, um carryout do bit mais significativo precisa ser somado ao resultado

□ exemplo: somar 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/>																	
contorna	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
<hr/>																	
soma	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	

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 transferência confiável de dados

REDES DE
COMPUTADORES
E A INTERNET 5ª edição

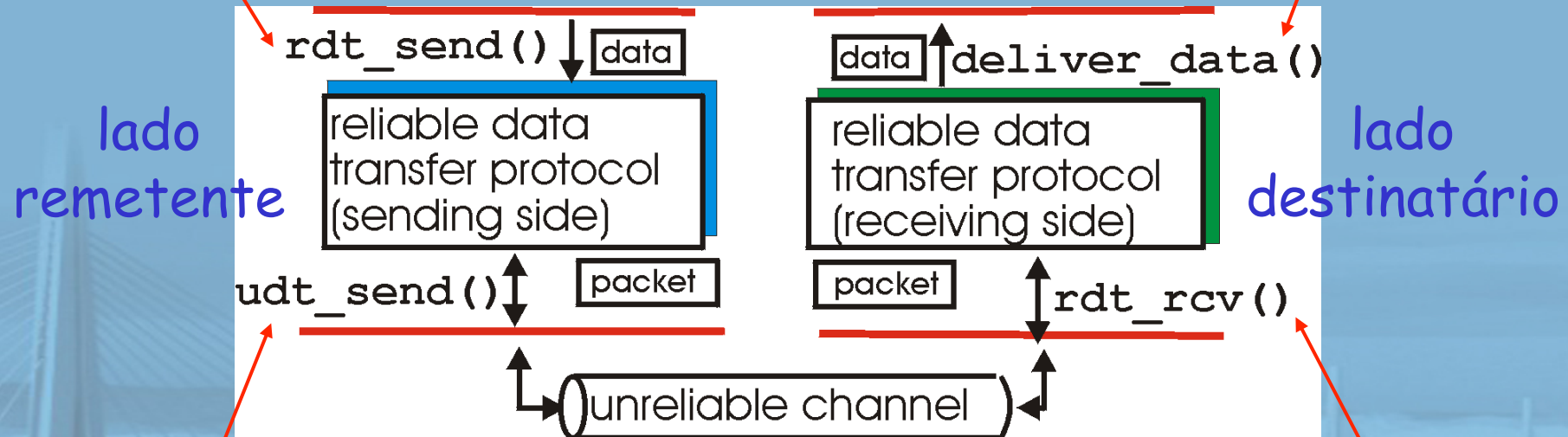
Uma Abordagem Top-Down

- ❑ importante nas camadas de aplicação, transporte e enlace
- ❑ lista dos 10 mais importantes tópicos de redes!
- ❑ características do canal confiável determinarão complexidade do protocolo de transferência confiável (rdt)

Transferência confiável de dados: introdução

rdt_send() : chamado de cima, (p. e., pela apl.). Dados passados para remeter à camada superior do destinatário

deliver_data() : chamado pela rdt para remeter dados para cima



udt_send() : chamado pela rdt, para transferir pacote por canal não confiável ao destinatário

rdt_rcv() : chamado quando pacote chega no lado destinatário do canal

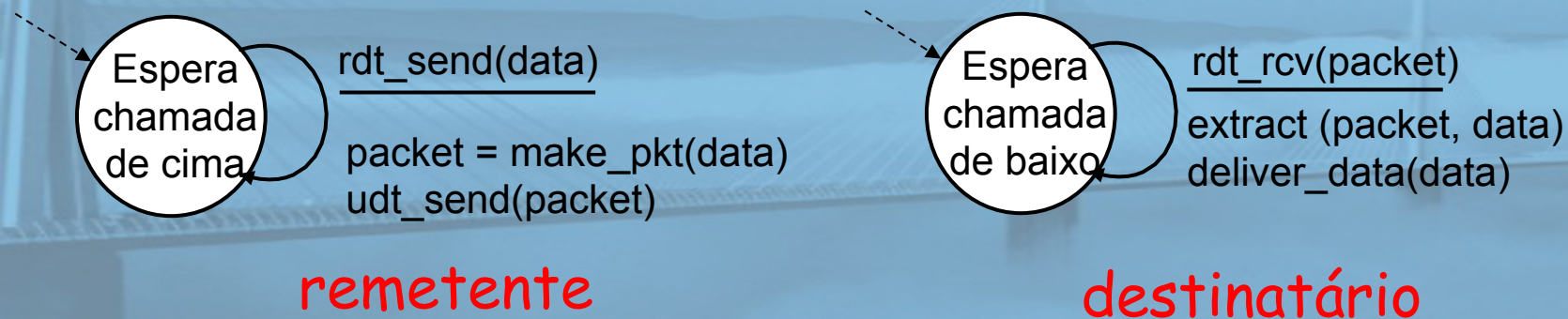
vamos:

- ❑ desenvolver de forma incremental os lados remetente e destinatário do protocolo de transferência confiável de dados (rdt)
- ❑ considerar apenas a transf. de dados unidirecional
 - mas informações de controle fluirão nas duas direções!
- ❑ usar máquinas de estado finito (FSM) para especificar remetente, destinatário



Rdt1.0: transferência confiável por canal confiável

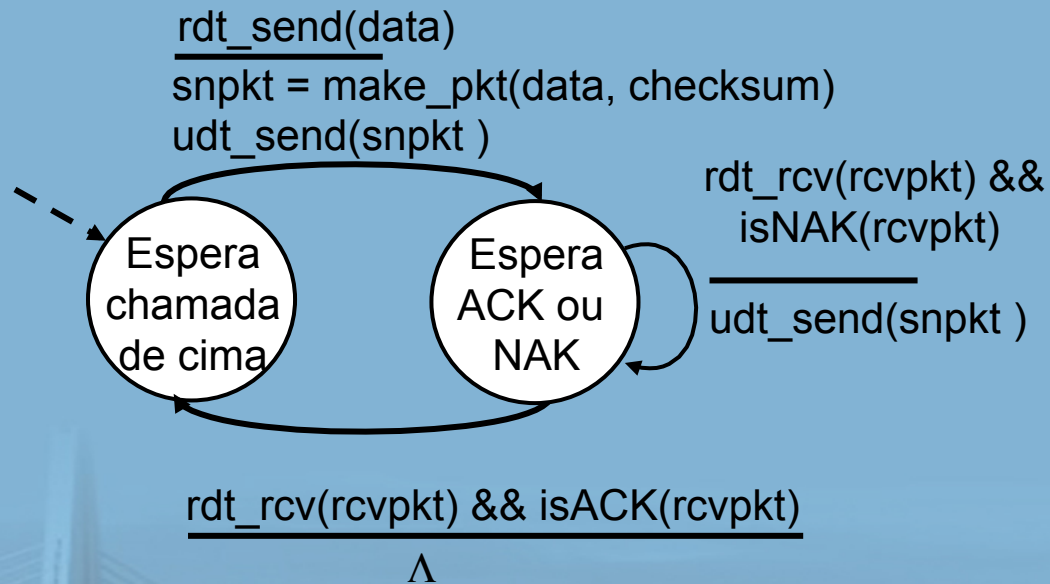
- ❑ canal subjacente perfeitamente confiável
 - sem erros de bit
 - sem perda de pacotes
 - destinatário não precisa informar o remetente
 - destinatário pode receber dados a qualquer taxa
- ❑ FSMs separadas para remetente e destinatário:
 - remetente envia dados para canal subjacente
 - destinatário lê dados do canal subjacente



Rdt2.0: canal com erros de bit

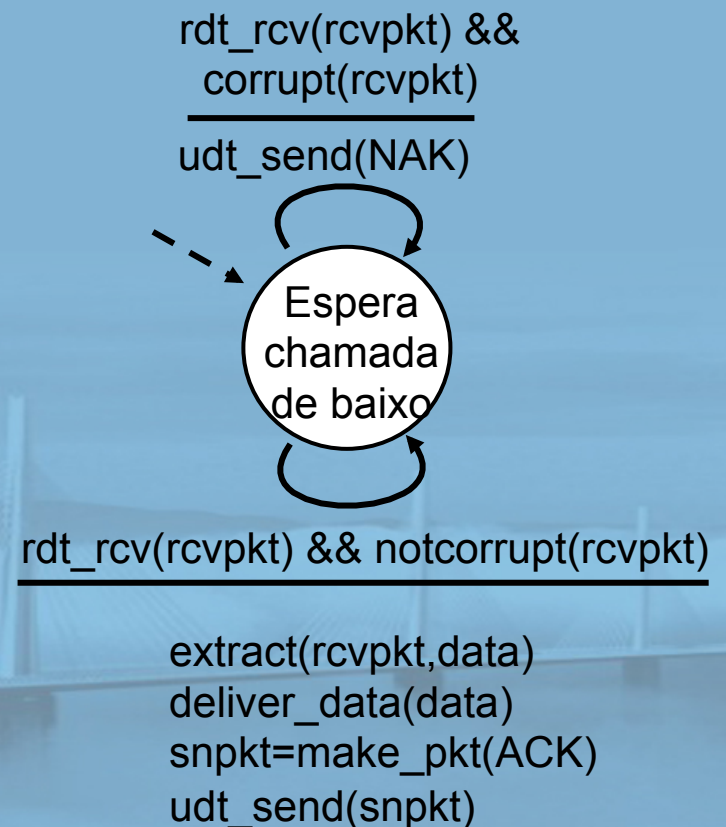
- ❑ canal subjacente pode inverter bits no pacote
 - soma de verificação para detectar erros de bit
- ❑ a questão: como recuperar-se dos erros:
 - *reconhecimentos (ACKs)*: destinatário diz explicitamente ao remetente que o pacote foi recebido OK
 - *reconhecimentos negativos (NAKs)*: destinatário diz explicitamente ao remetente que o pacote teve erros
 - remetente retransmite pacote ao receber NAK
- ❑ novos mecanismos no rdt2.0 (além do rdt1.0):
 - detecção de erro
 - feedback do destinatário: msgs de controle (ACK,NAK) destinatário->remetente

rdt2.0: especificação da FSM

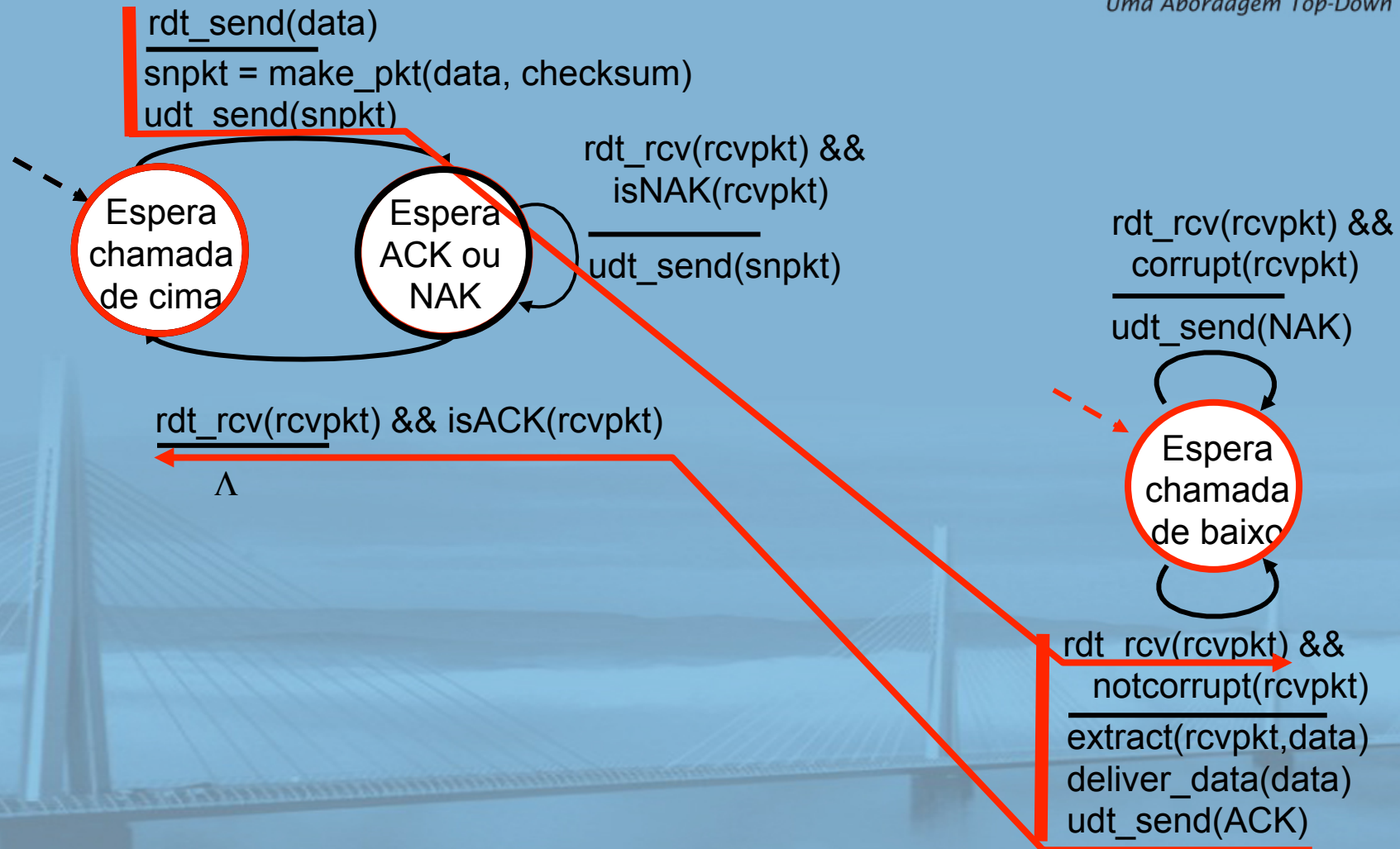


remetente

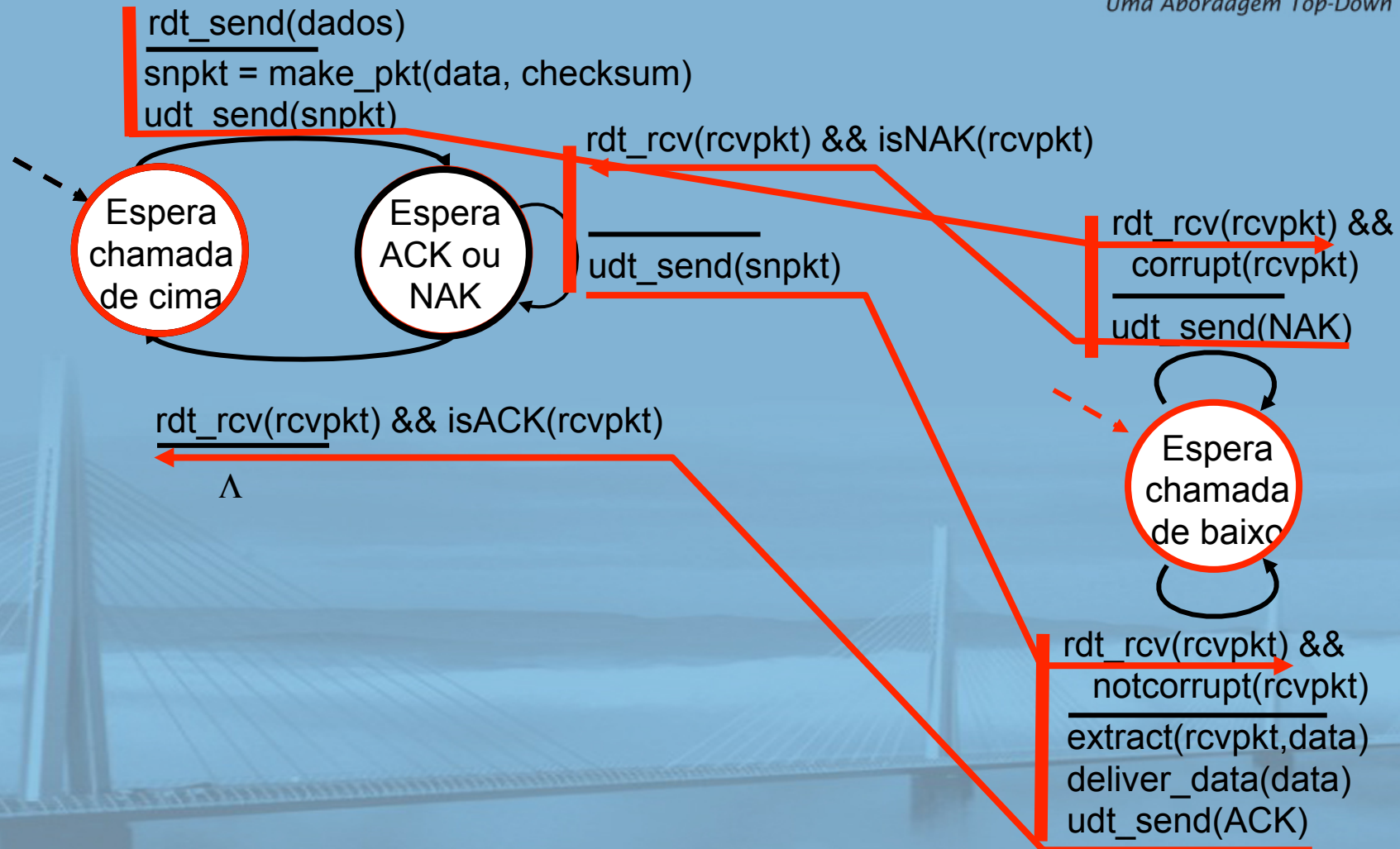
destinatário



rdt2.0: operação sem erros



rdt2.0: cenário de erro



rdt2.0 tem uma falha fatal!

O que acontece se ACK/ NAK for corrompido?

- ❑ remetente não sabe o que aconteceu no destinatário!
- ❑ não pode simplesmente retransmitir: possível duplicação

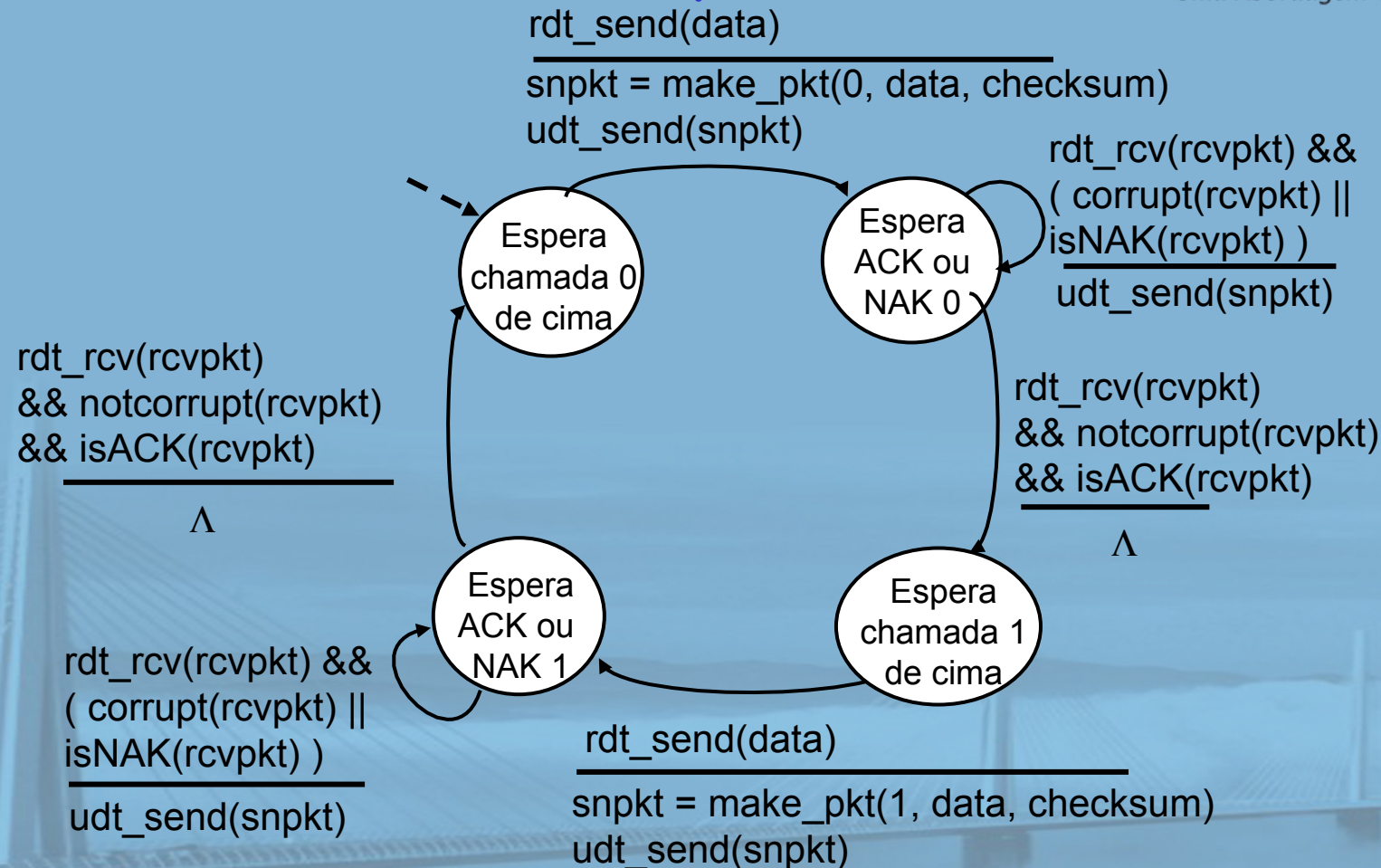
tratando de duplicatas:

- ❑ remetente retransmite pacote atual se ACK/NAK corrompido
- ❑ remetente acrescenta *número de sequência* a cada pacote
- ❑ destinatário descarta (não sobe) pacote duplicado

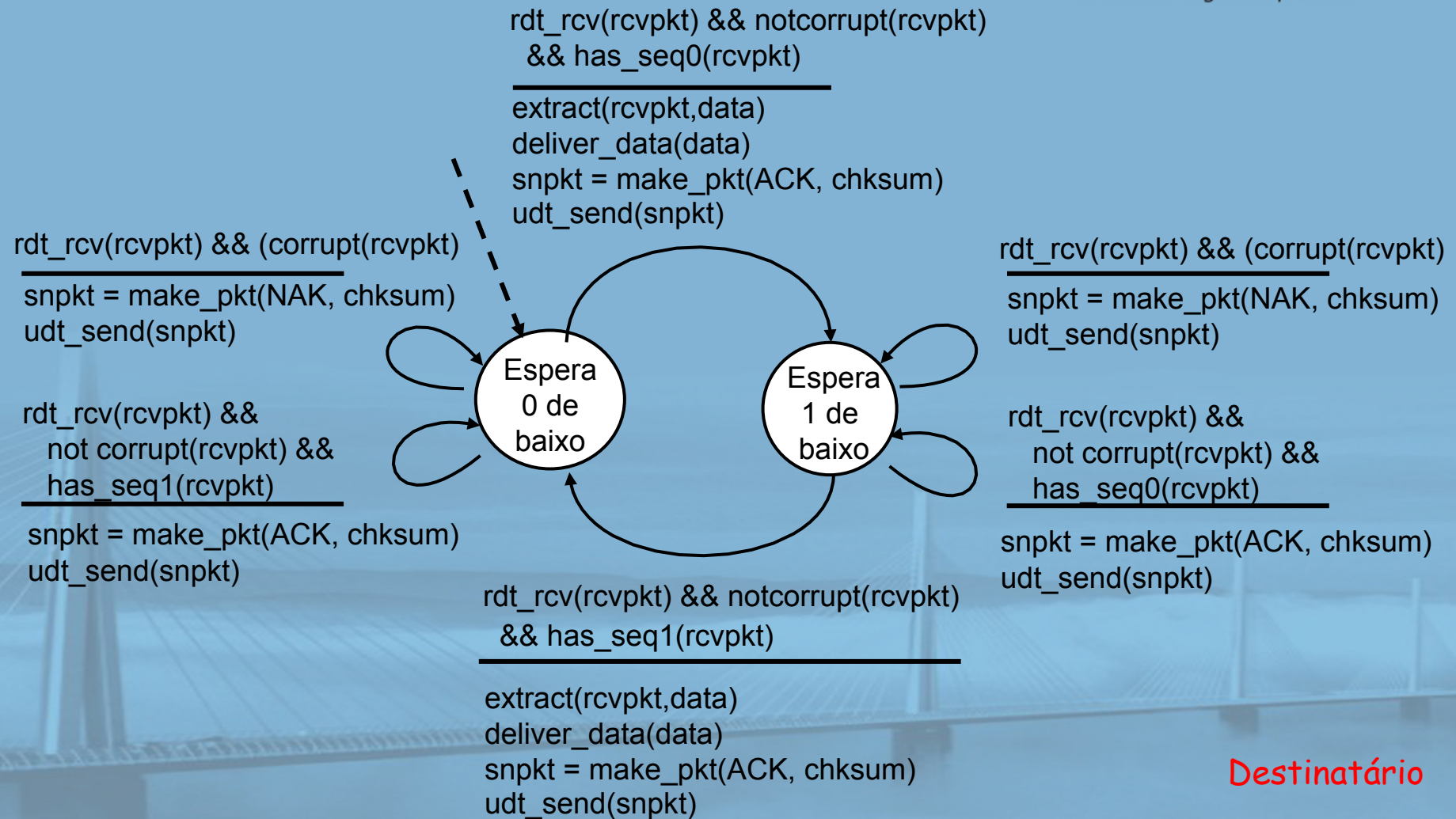
pare e espere

remetente envia um pacote, depois espera resposta do destinatário

rdt2.1: remetente trata de ACK/NAKs corrompidos



Remetente



rdt2.1: discussão

remetente:

- ❑ # seq acrescentado ao pkt
- ❑ dois #s seq. (0,1) bastarão. Por quê?
- ❑ deve verificar se ACK/NAK recebido foi corrompido
- ❑ o dobro de estados
 - estado de "lembrar" se pacote "atual" tem # seq. 0 ou 1

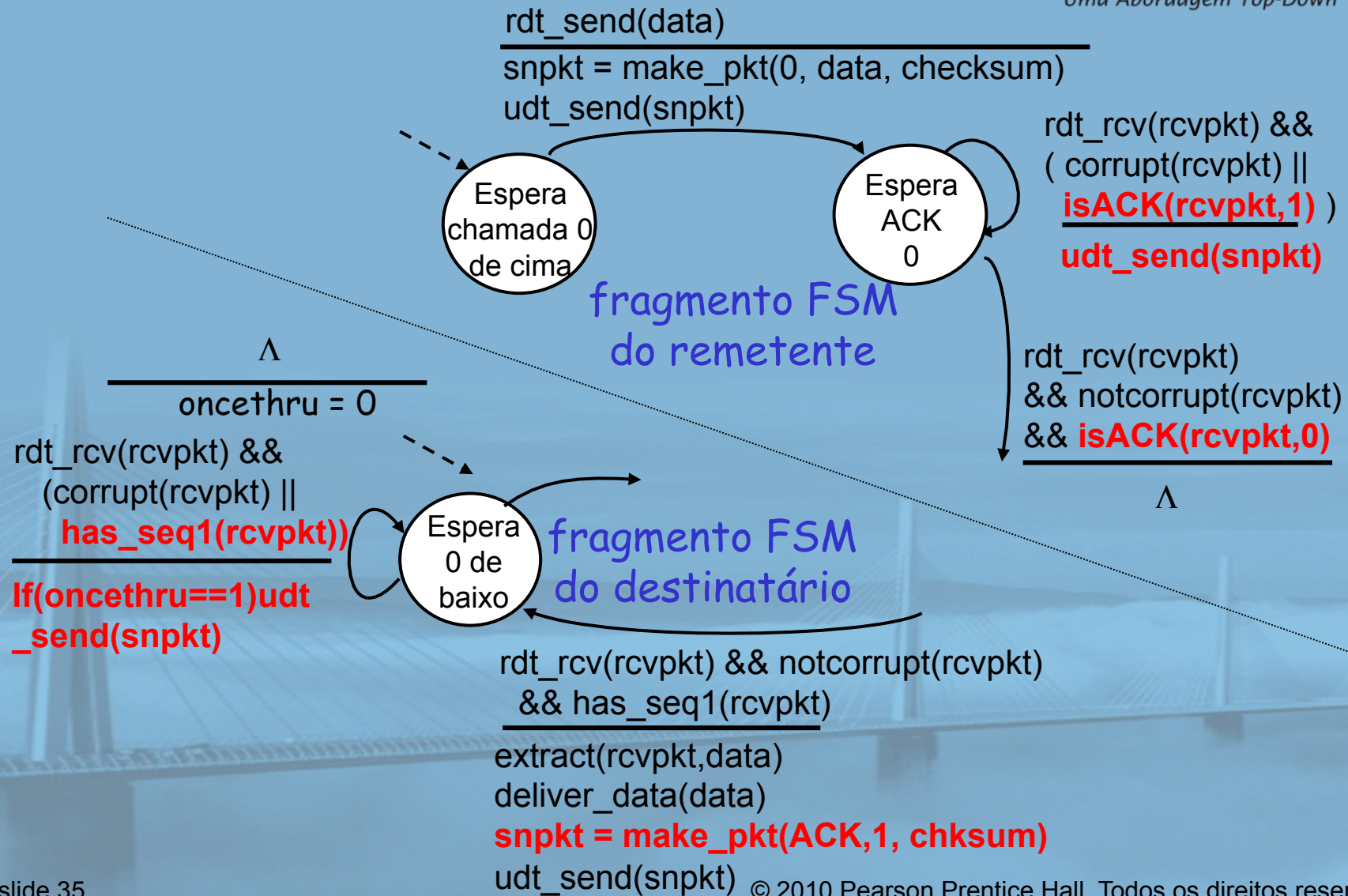
destinatário:

- ❑ deve verificar se pacote recebido está duplicado
 - estado indica se 0 ou 1 é # seq. esperado do pacote
- ❑ nota: destinatário *não* sabe se seu último ACK/NAK foi recebido OK no remetente

rdt2.2: um protocolo sem NAK

- ❑ mesma funcionalidade de rdt2.1, usando apenas ACKs
- ❑ em vez de NAK, destinatário envia ACK para último pacote recebido OK
 - destinatário precisa incluir *explicitamente* # seq. do pacote sendo reconhecido com ACK
- ❑ ACK duplicado no remetente resulta na mesma ação de NAK: *retransmitir pacote atual*

rdt2.2: fragmentos do remetente, destinatário



rdt3.0: canais com erros e perda

nova suposição: canal subjacente também pode perder pacotes (dados ou ACKs)

- soma de verificação, # seq., ACKs, retransmissões serão úteis, mas não suficientes

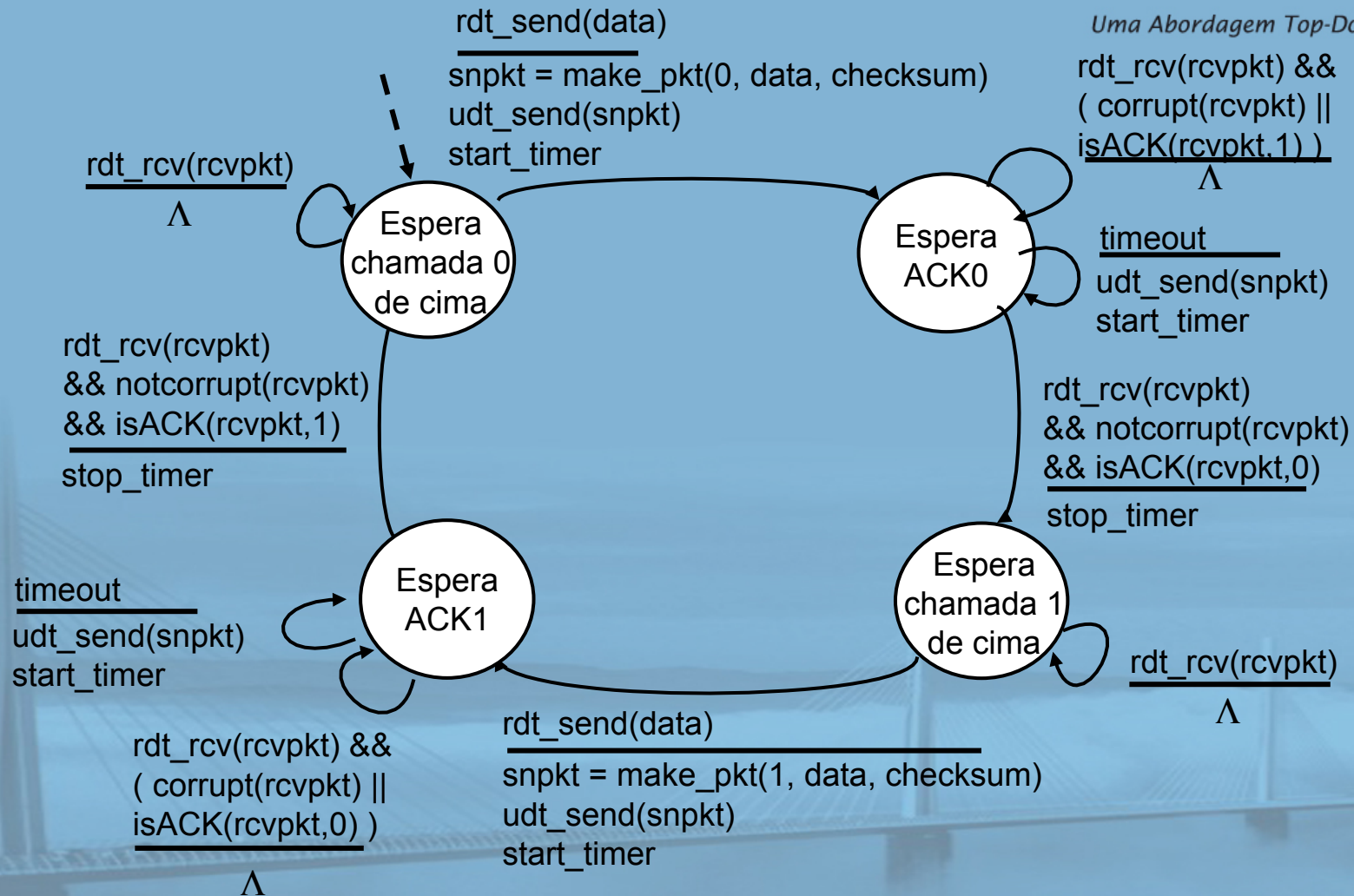
técnica: remetente espera quantidade "razoável" de tempo por ACK

- retransmite se não chegar ACK nesse tempo
- se pct (ou ACK) simplesmente atrasado (não perdido):
 - retransmissão será duplicada, mas os #s de seq. já cuidam disso
 - destinatário deve especificar # seq. do pacote sendo reconhecido com ACK
- requer contador regressivo

remetente rdt3.0

REDES DE COMPUTADORES E A INTERNET 5ª edição

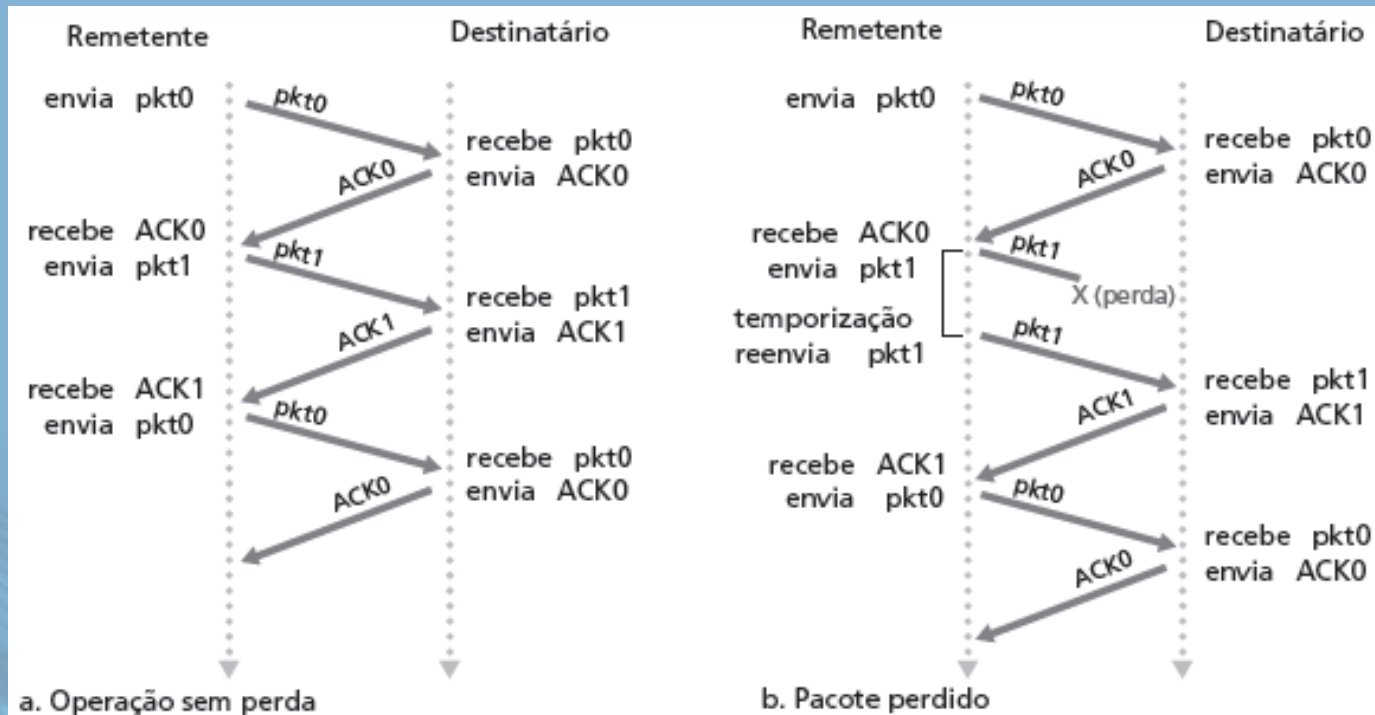
Uma Abordagem Top-Down



rdt3.0 em ação

REDES DE COMPUTADORES E A INTERNET 5ª edição

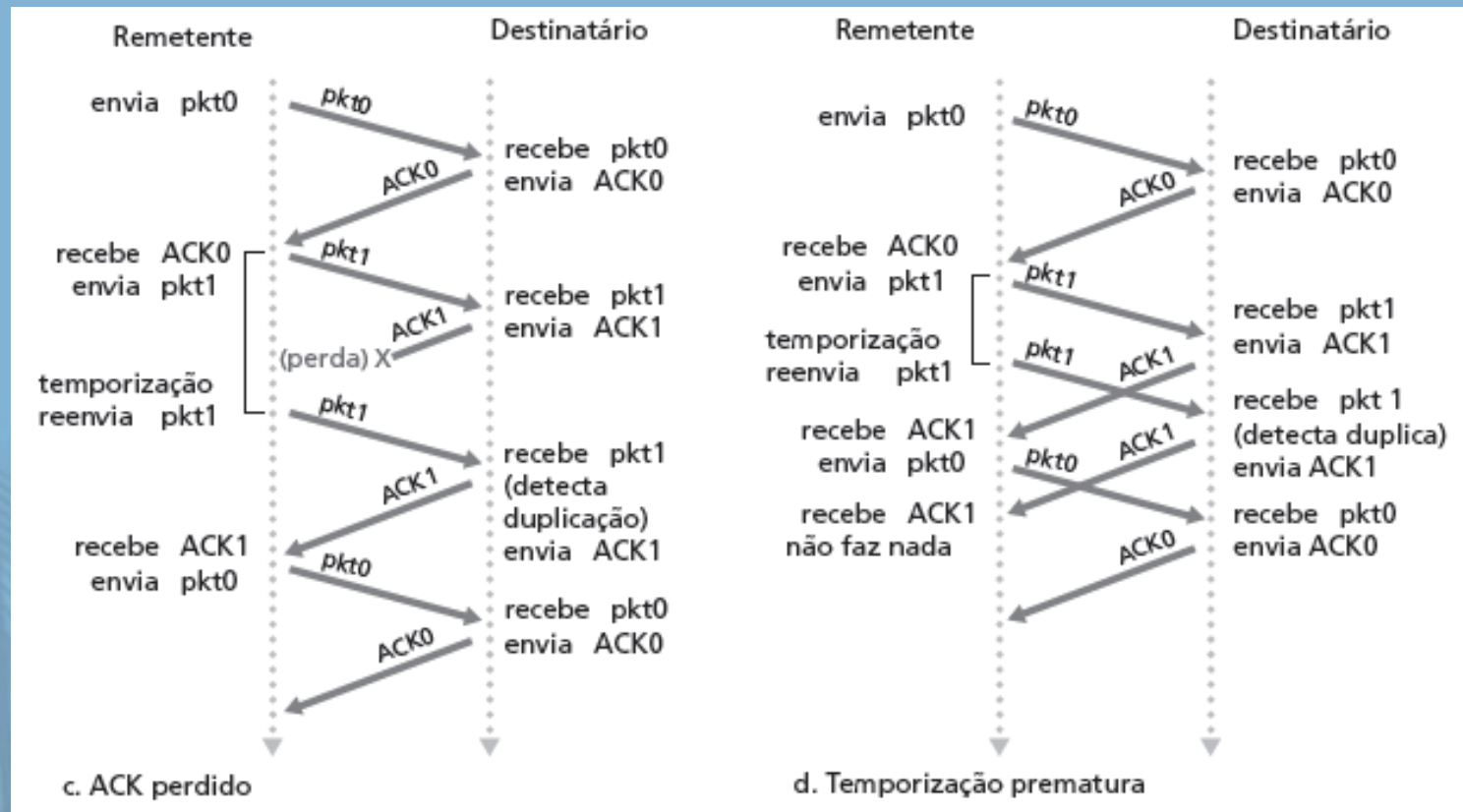
Uma Abordagem Top-Down



REDES DE COMPUTADORES E A INTERNET

5ª edição

Uma Abordagem Top-Down



Desempenho do rdt3.0

- ❑ rdt3.0 funciona, mas com desempenho ruim
- ❑ ex.: enlace 1 Gbps, 30 ms atraso propagação, pacote 8000 bits:

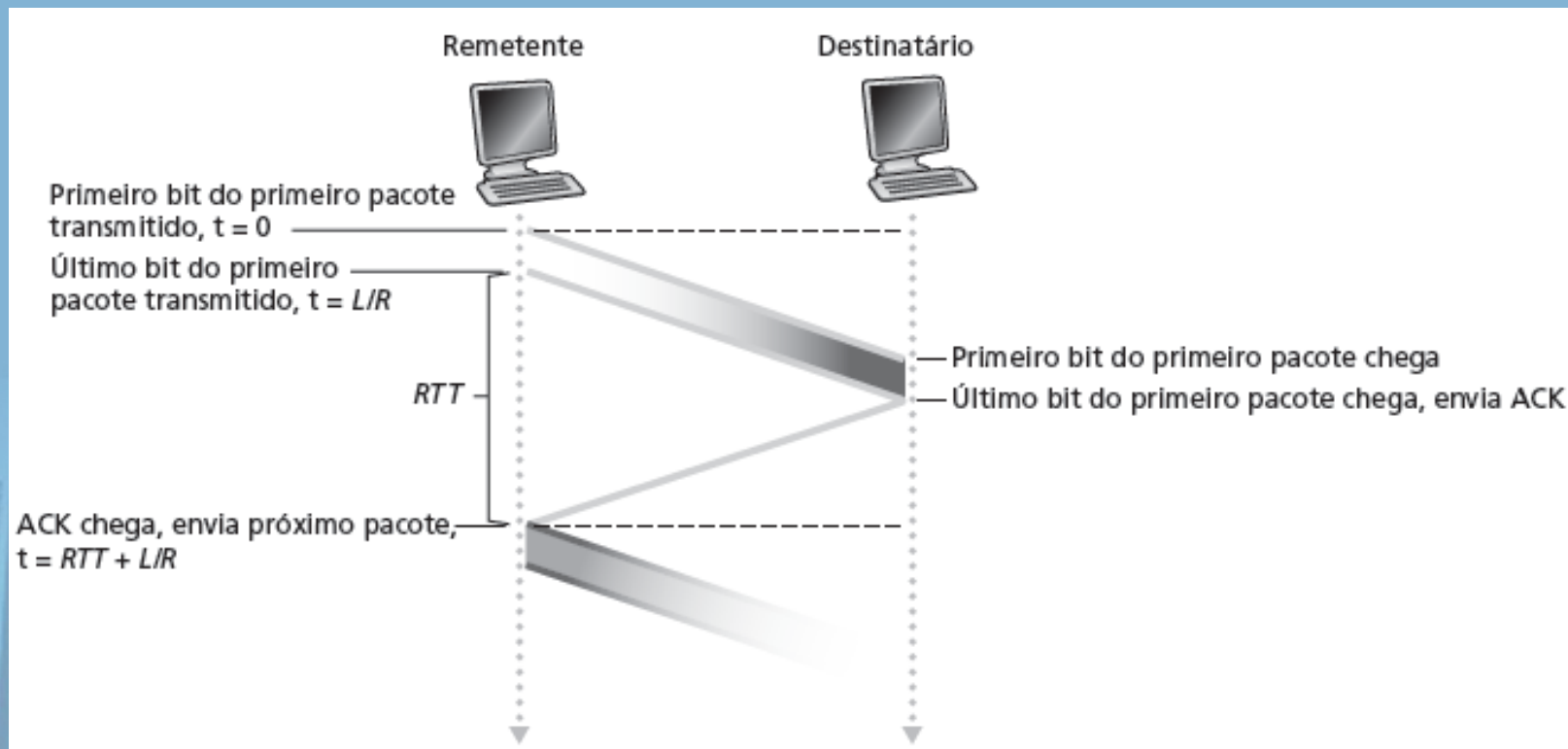
$$t_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{bps}} = 8\text{microssegundos}$$

- U_{remet} : **utilização** - fração do tempo remet. ocupado enviando

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

- Pct. 1 KB cada 30 ms -> 33 kB/s (267 Kbps) vazão em enlace de 1 Gbps
- protocolo de rede limita uso de recursos físicos!

rdt3.0: operação pare e espere

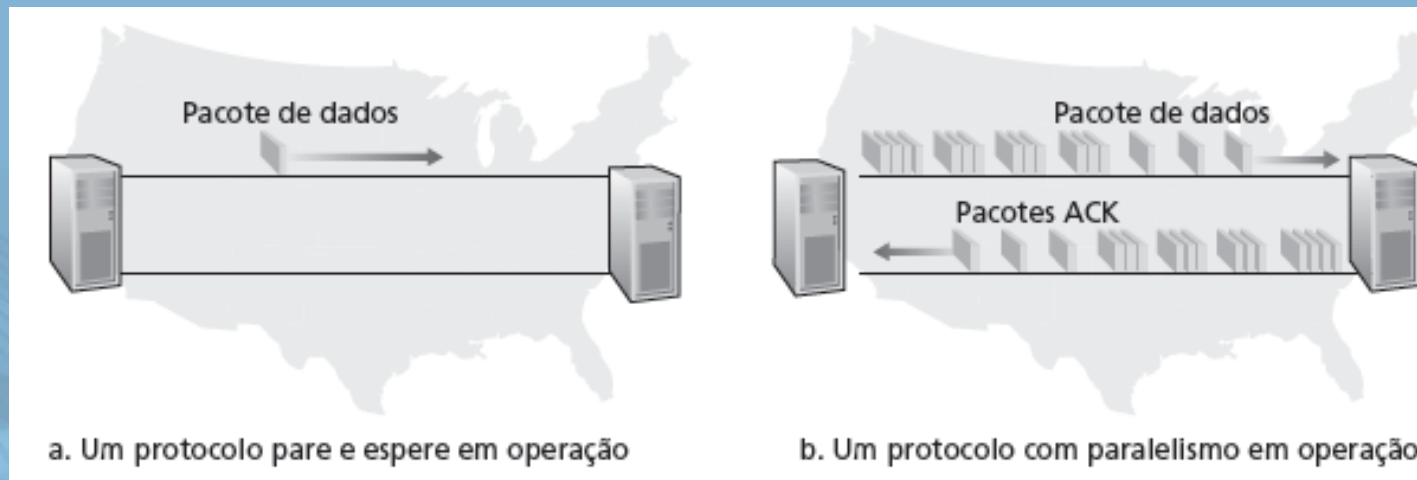


$$U_{\text{remet}} = \frac{L/R}{RTT + L/R} = \frac{0,008}{30,008} = 0,00027$$

Protocolos com paralelismo

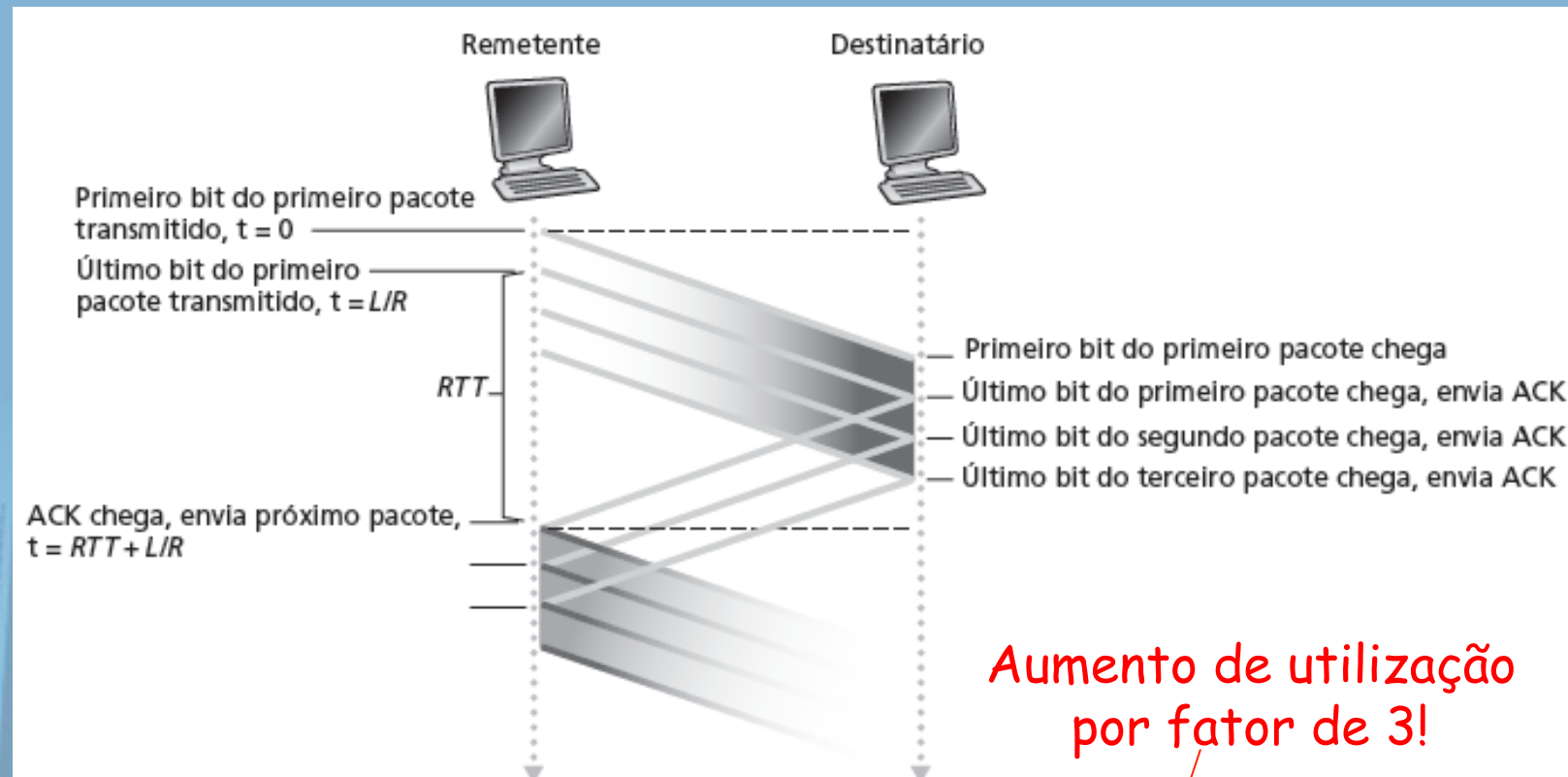
paralelismo: remetente permite múltiplos pacotes “no ar”, ainda a serem reconhecidos

- intervalo de números de sequência deve ser aumentado
- buffering no remetente e/ou destinatário



- duas formas genéricas de protocolo com paralelismo:
Go-Back-N, repetição seletiva

Paralelismo: utilização aumentada



$$U_{\text{remet}} = \frac{3 * L / R}{RTT + L / R} = \frac{0,02}{30,008} = 0,0008$$

Protocolos com paralelismo

Go-back-N: visão geral

- ❑ *remetente*: até N pacotes não reconhecidos na pipeline
- ❑ *destinatário*: envia ACKs cumulativos
 - não envia pct ACK se houver uma lacuna
- ❑ *remetente*: tem temporizador para pct sem ACK mais antigo
 - se o temporizador expirar: retransmite todos os pacotes sem ACK

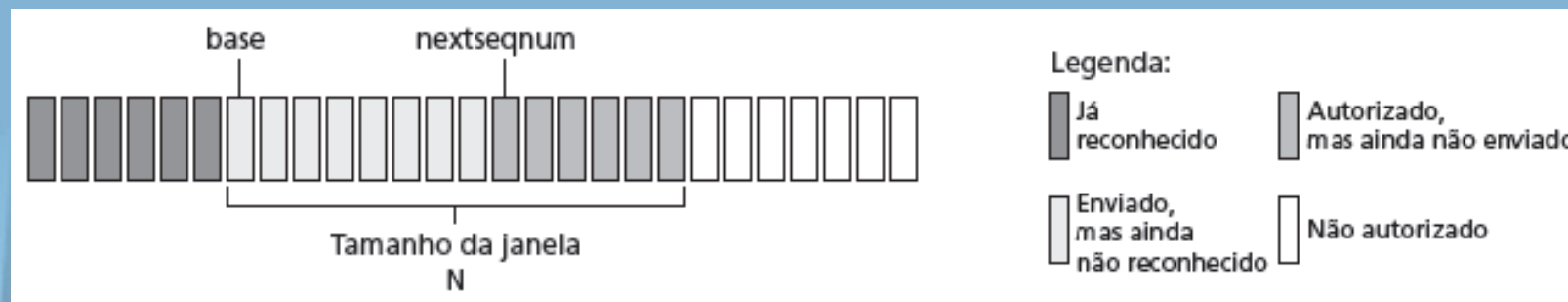
Repetição seletiva: visão geral

- ❑ *remetente*: até N pacotes não reconhecidos na pipeline
- ❑ *destinatário*: reconhece (ACK) pacotes individuais
- ❑ *remetente*: mantém temporizador para cada pct sem ACK
 - se o temporizador expirar: retransmite apenas o pacote sem ACK

Go-Back-N

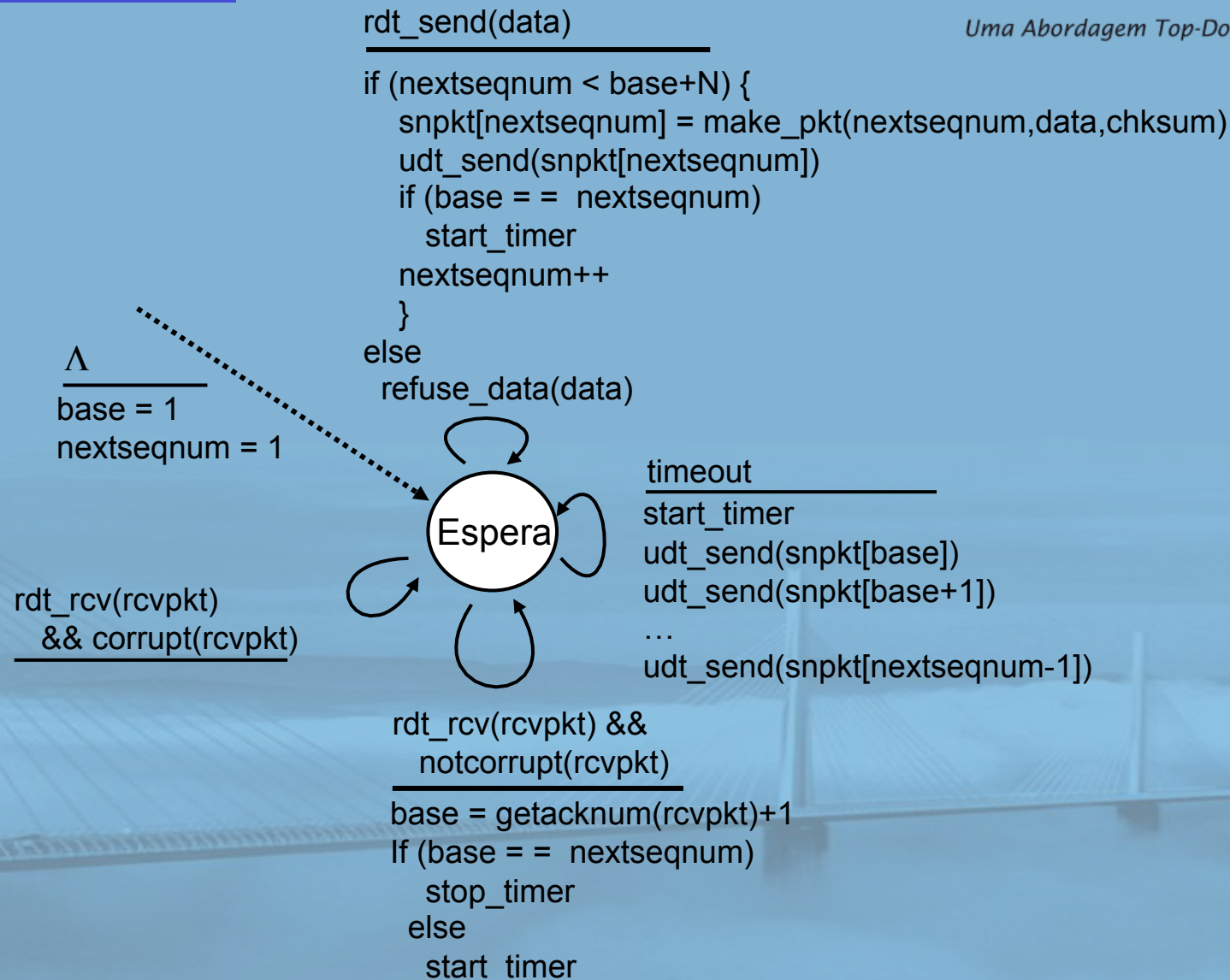
remetente:

- ❑ # seq. de k bits no cabeçalho do pacote
- ❑ "janela" de até N pcts consecutivos sem ACK permitidos

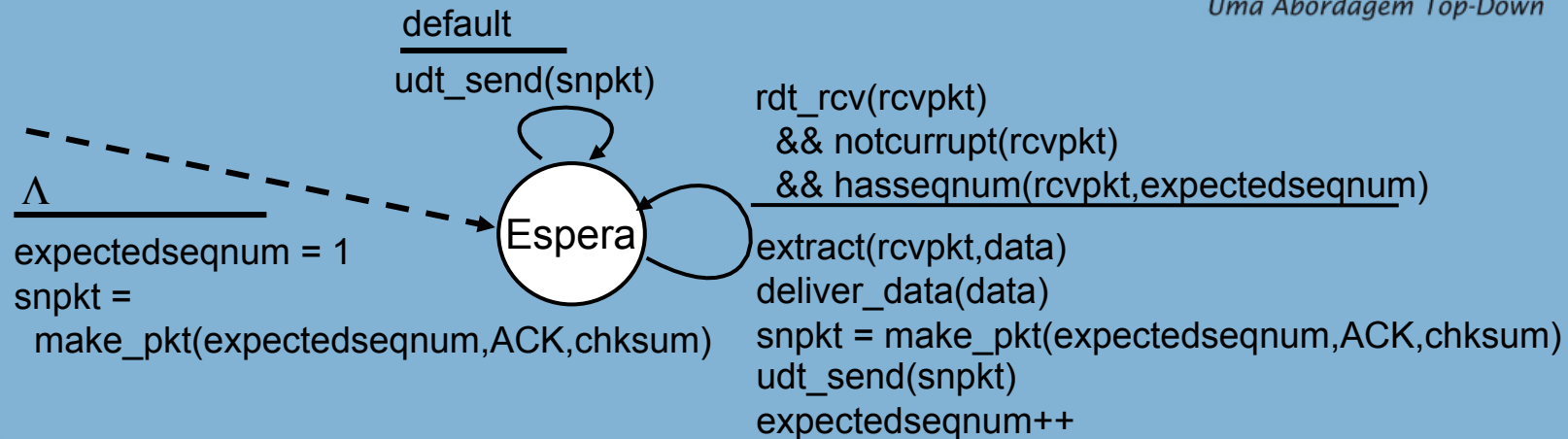


- ❑ ACK(n): ACK de todos pcts até inclusive # seq. n - "ACK cumulativo"
 - pode receber ACKs duplicados (ver destinatário)
- ❑ temporizador para cada pacote no ar
- ❑ *timeout(n)*: retransmite pct n e todos pcts com # seq. mais alto na janela

GBN: FSM estendido no remetente



GBN: FSM estendido no destinatário



apenas ACK: sempre envia ACK para pct recebido corretamente com # seq. mais alto *em ordem*

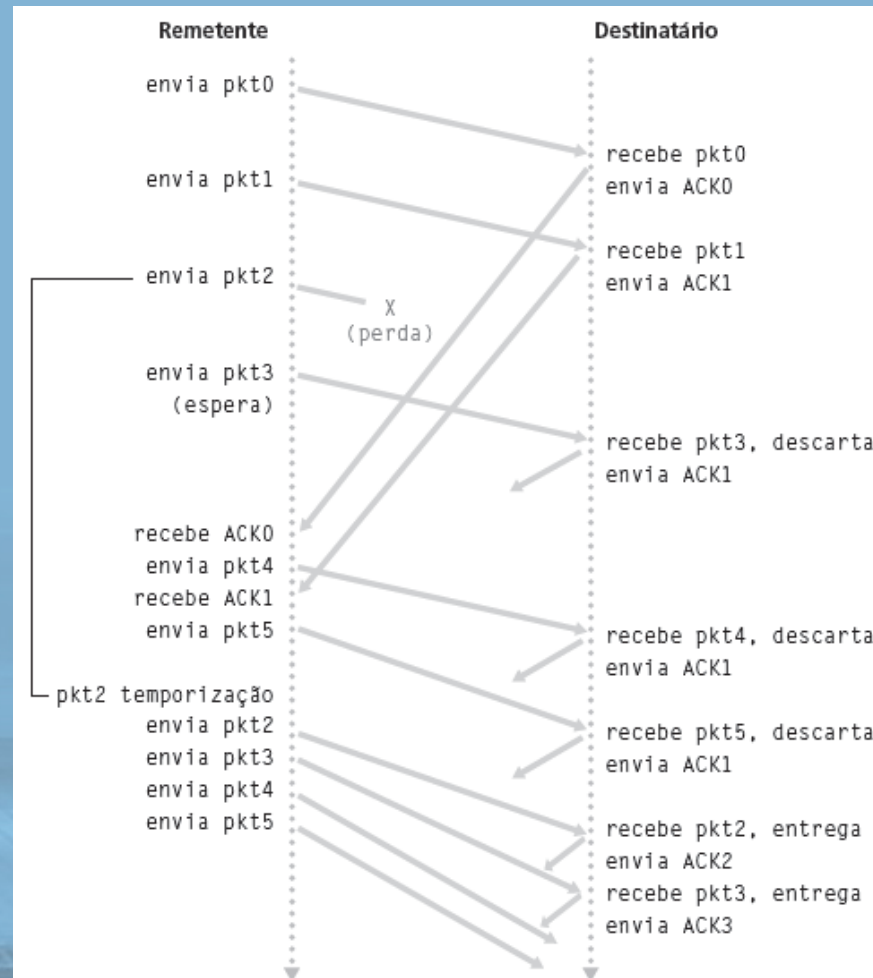
- pode gerar ACKs duplicados
- só precisa se lembrar de **expectedseqnum**

□ pacote fora de ordem:

- descarta (não mantém em buffer) -> **sem buffering no destinatário!**
- reenvia ACK do pct com # seq. mais alto em ordem

GBN em operação

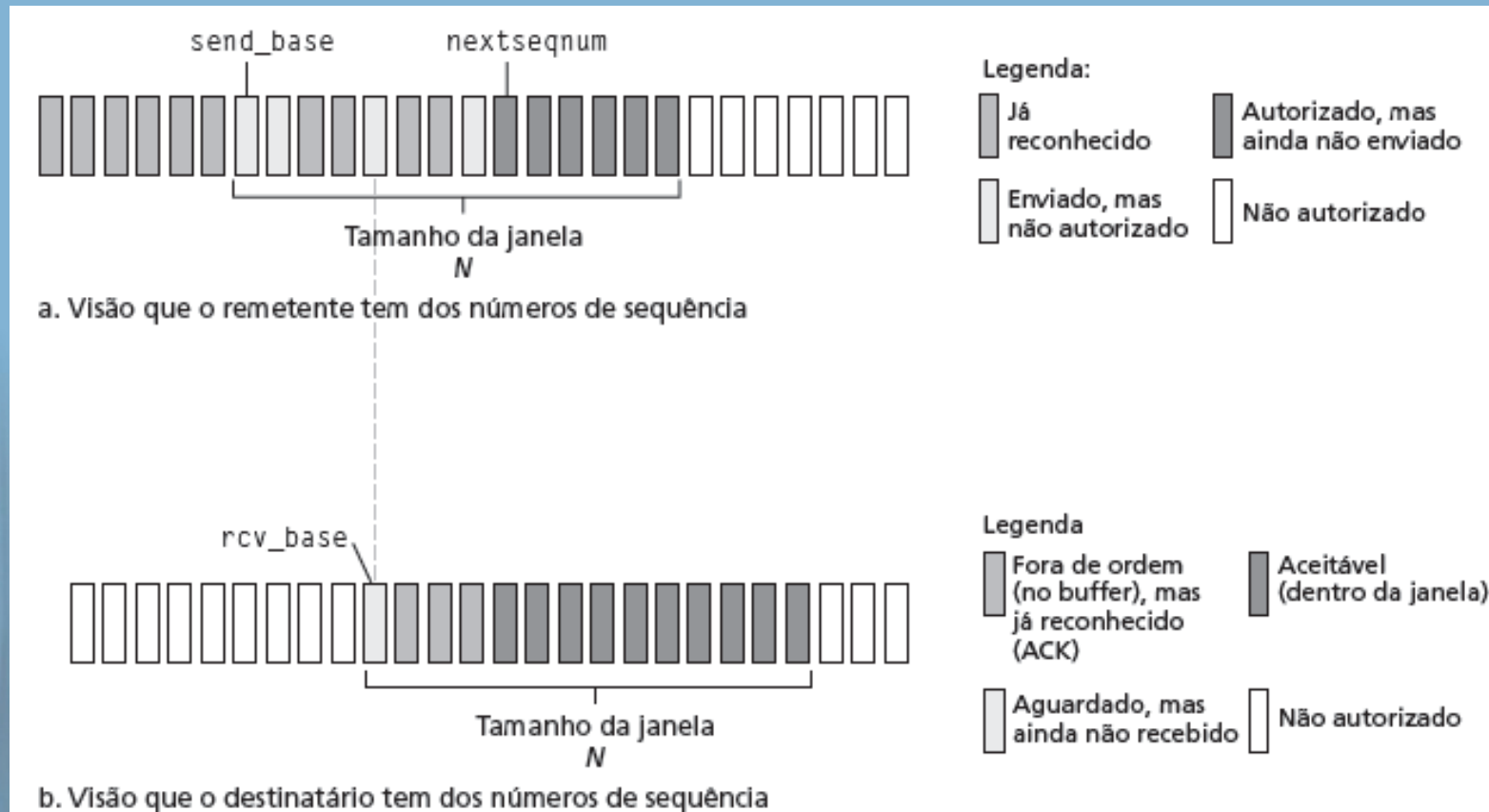
Uma Abordagem Top-Down



Repetição seletiva

- ❑ destinatário reconhece *individualmente* todos os pacotes recebidos de modo correto
 - mantém pcts em buffer, se for preciso, para eventual remessa em ordem para a camada superior
- ❑ remetente só reenvia pcts para os quais o ACK não foi recebido
 - temporizador no remetente para cada pct sem ACK
- ❑ janela do remetente
 - N # seq. consecutivos
 - novamente limita #s seq. de pcts enviados, sem ACK

Repetição seletiva: janelas de remetente, destinatário



Repetição seletiva

remetente

dados de cima:

- ❑ se próx. # seq. disponível na janela, envia pct

timeout(n):

- ❑ reenvia pct n, reinicia temporizador

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

- ❑ marca pct n como recebido
- ❑ se n menor pct com ACK, avança base da janela para próximo # seq. sem ACK

destinatário

pct n em [rcvbase, rcvbase + N - 1]

- ❑ envia ACK(n)
- ❑ fora de ordem: buffer
- ❑ em ordem: entrega (também entrega pcts em ordem no buffer), avança janela para próximo pct ainda não recebido

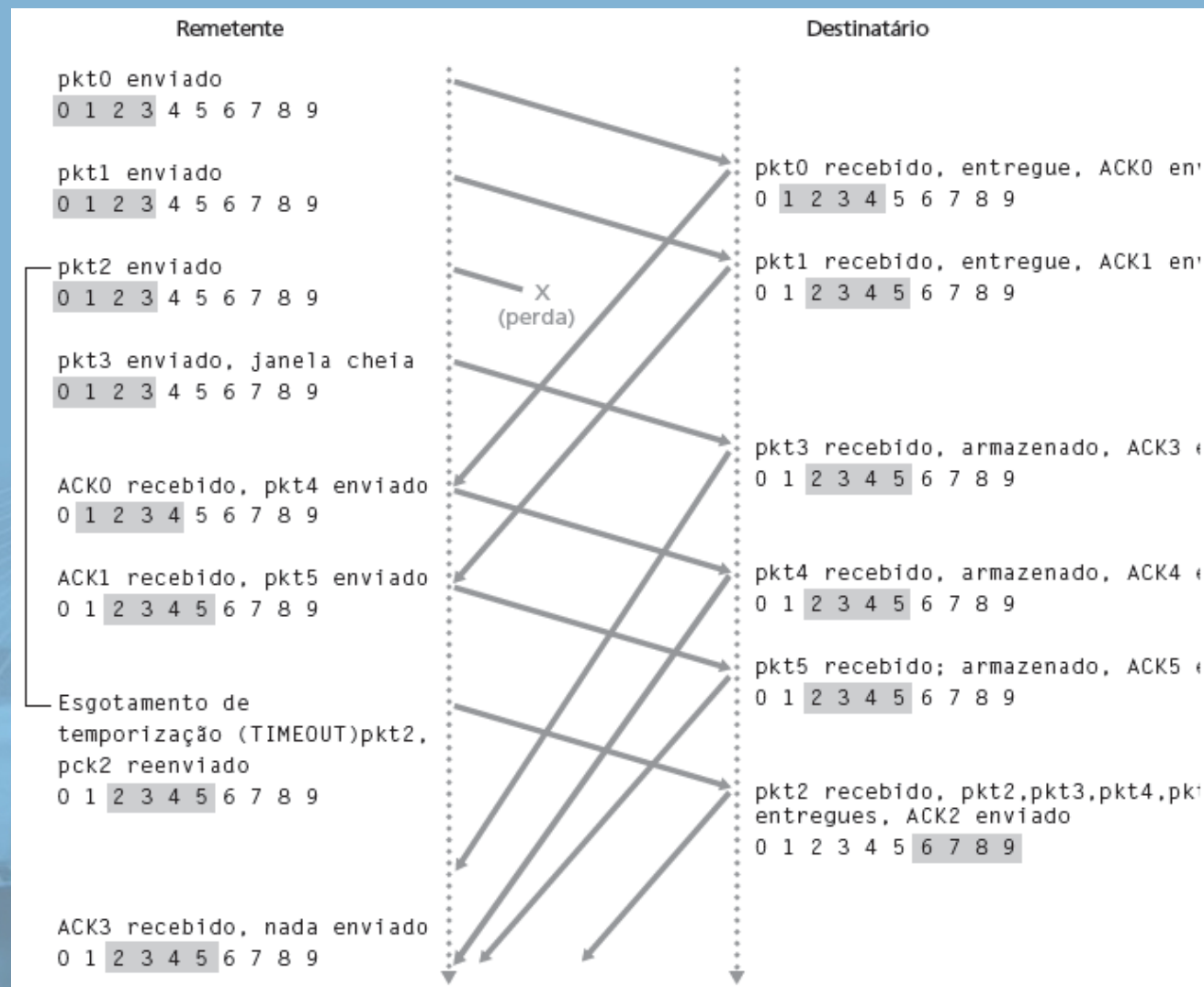
pct n em [rcvbase - N, rcvbase - 1]

- ❑ ACK(n)

caso contrário:

- ❑ ignora

Repetição seletiva em operação



Repetição seletiva: dilema

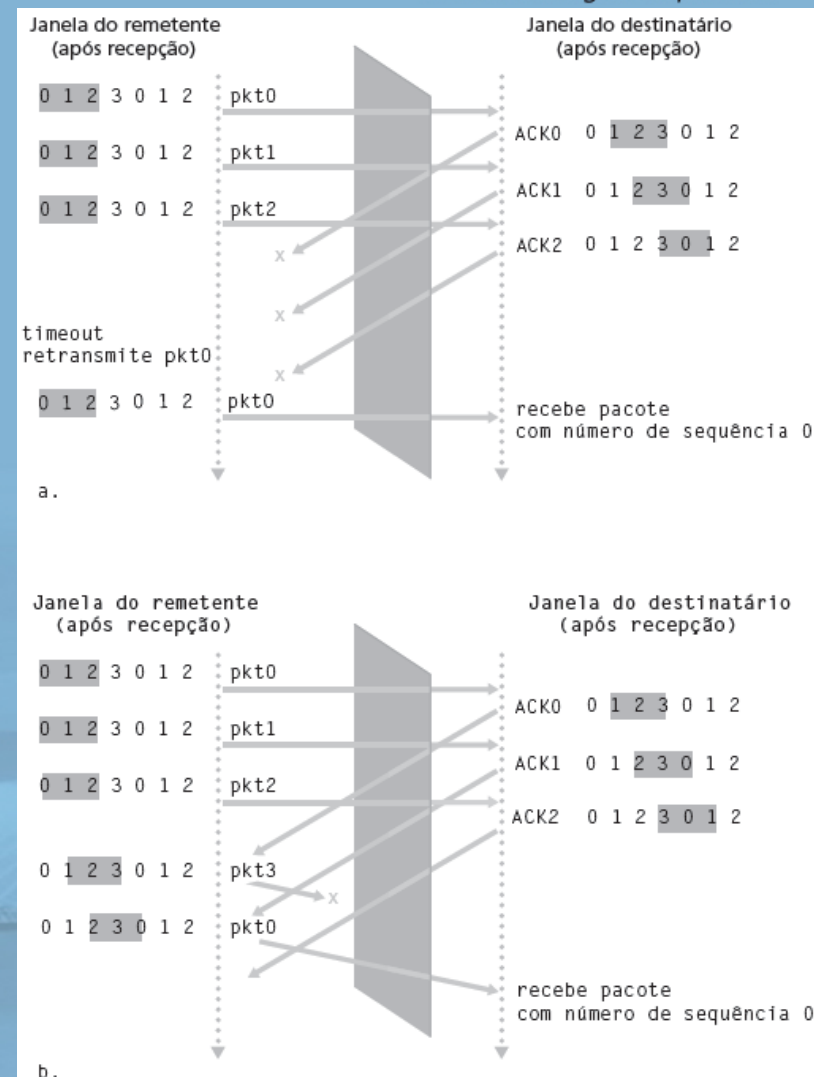
Exemplo:

- ❑ # seq.: 0, 1, 2, 3
- ❑ tamanho janela = 3
- ❑ destinatário não vê diferença nos dois cenários!
- ❑ passa incorretamente dados duplicados como novos em (a)

P: Qual o relacionamento entre tamanho do # seq. e tamanho de janela?

REDES DE COMPUTADORES E A INTERNET 5ª edição

Uma Abordagem Top-Down



Capítulo 3: Esboço

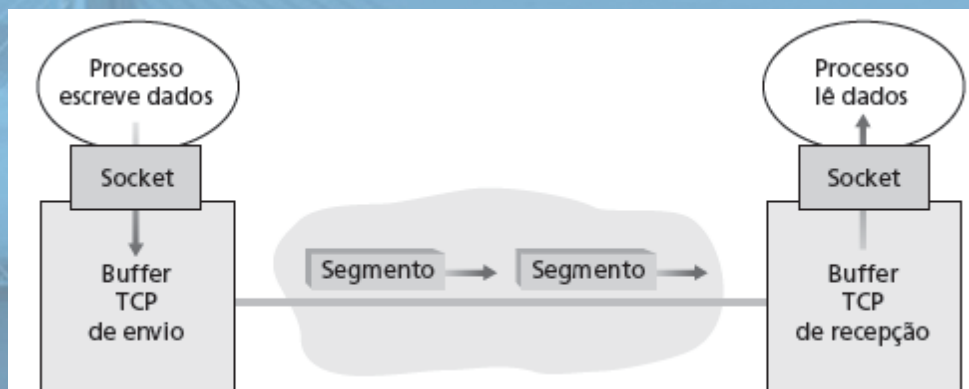
- ❑ 3.1 Serviços da camada de transporte
- ❑ 3.2 Multiplexação e demultiplexação
- ❑ 3.3 Transporte não orientado para conexão: UDP
- ❑ 3.4 Princípios da transferência confiável de dados
- ❑ 3.5 Transporte orientado para conexão: TCP
 - estrutura de segmento
 - transferência confiável de dados
 - controle 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

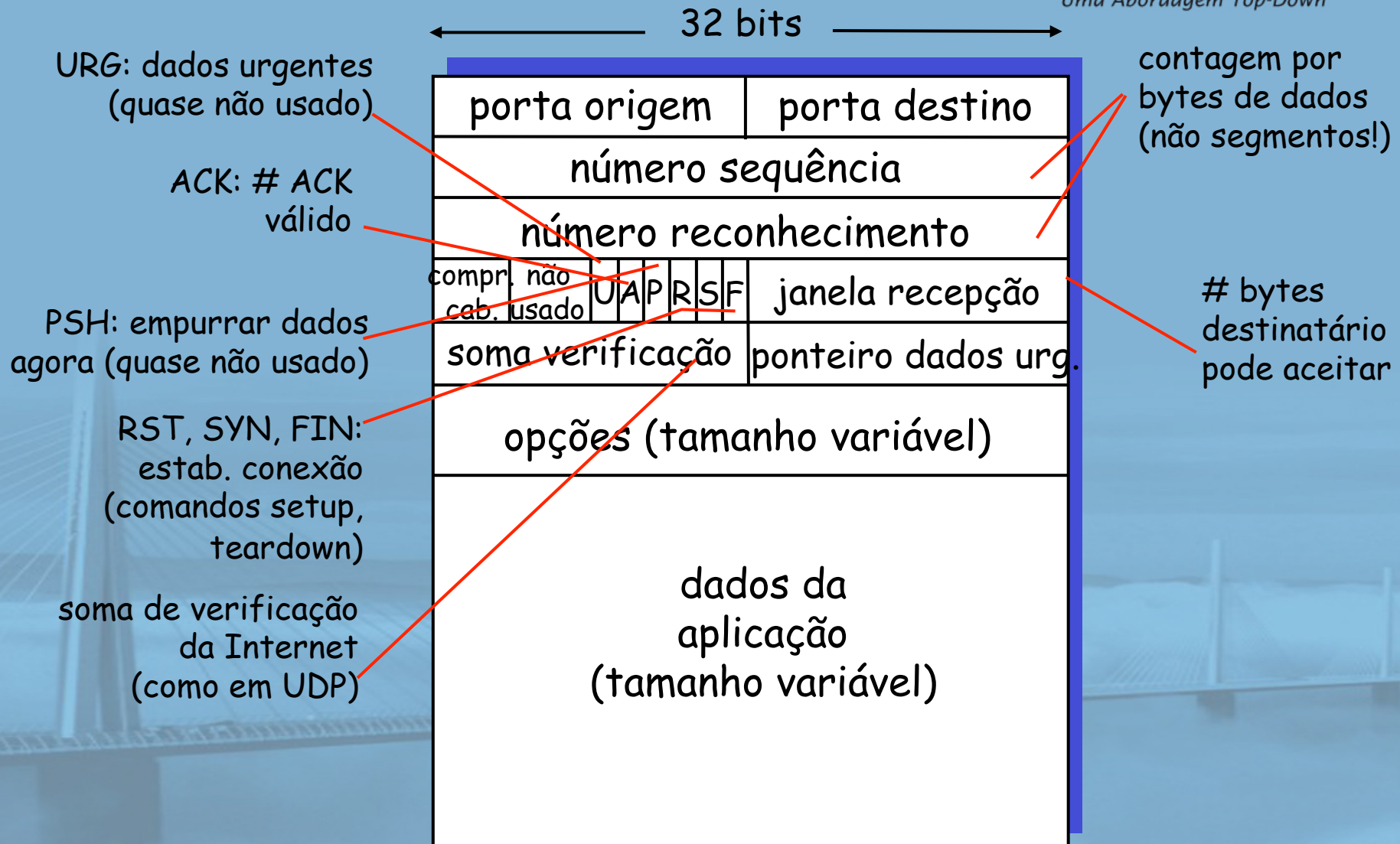
- ❑ ponto a ponto (unicast), um destinatário
- ❑ *cadeia de bytes confiável, em ordem*
- ❑ *paralelismo:*
 - congestionamento TCP e controle de fluxo definem tamanho da janela
- ❑ *buffers de envio & recepção*

- ❑ *dados full duplex:*
 - dados bidirecionais fluem na mesma conexão
 - MSS: tamanho máximo do segmento
- ❑ *orientado a conexão:*
 - apresentação (troca de msgs de controle), inicia variáveis de estado no remetente e destinatário antes da troca de dados
- ❑ *fluxo controlado:*
 - remetente não sobrecarrega destinatário



Estrutura do segmento TCP

Uma Abordagem Top-Down



#s sequência e ACKs do TCP

#s de sequência:

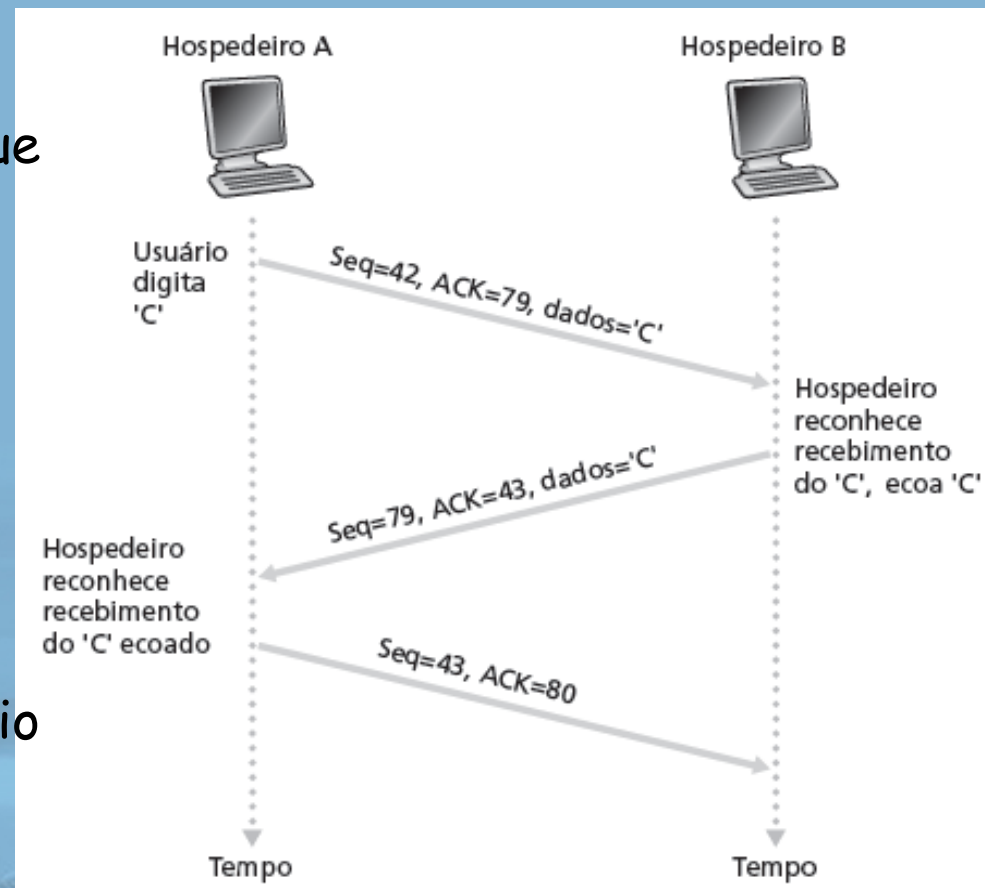
- representam os bytes que foram enviados

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 à 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

Média mantida pelo TCP



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

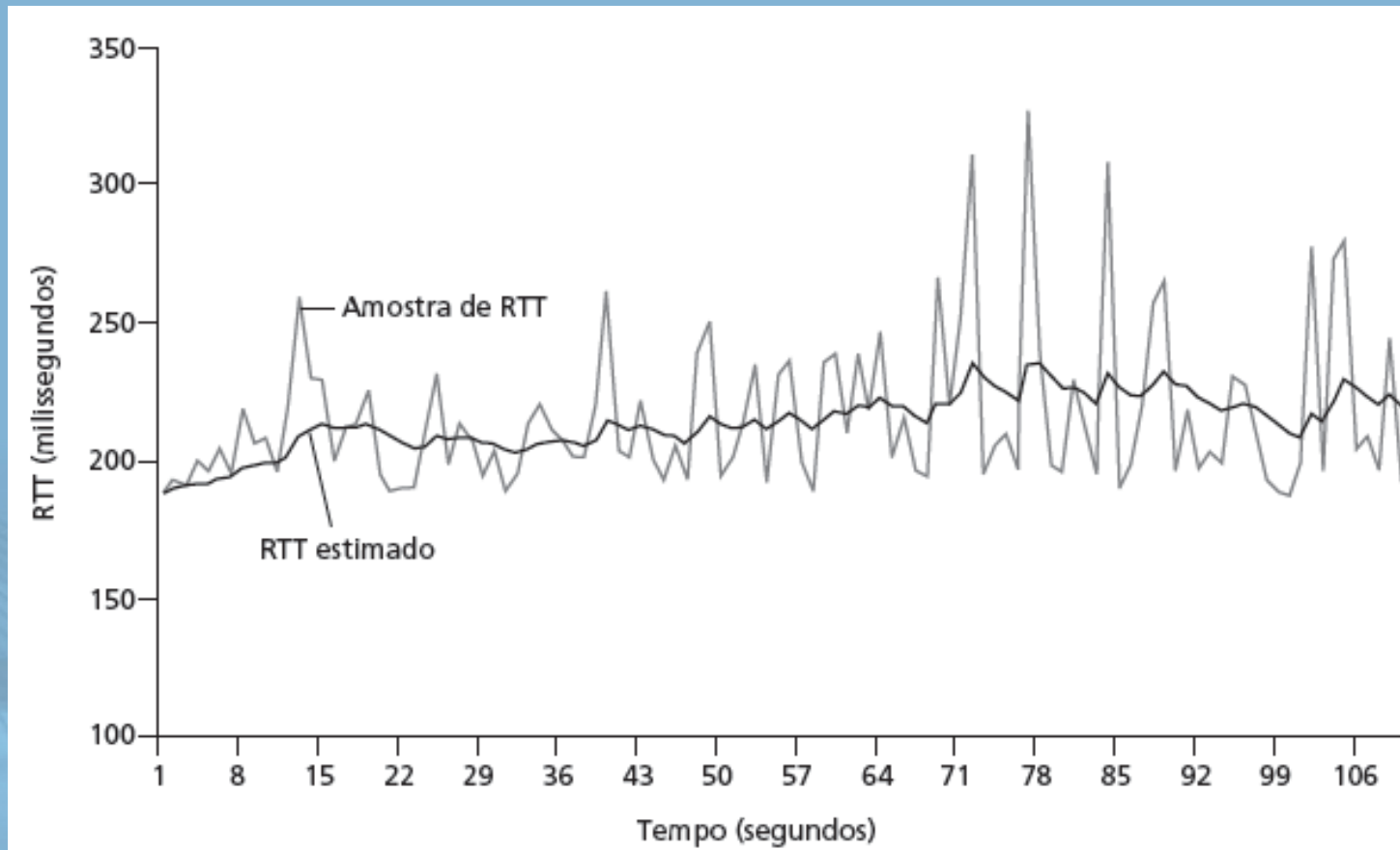
- ❑ combinação ponderada entre o EstimatedRTT anterior e o novo valor para SampleRTT
- ❑ média móvel exponencial ponderada (termo estatístico)
- ❑ influência da amostra passada diminui exponencialmente
- ❑ valor típico: $\alpha = 0,125$

$$\text{EstimatedRTT} = 0,875 * \text{EstimatedRTT} + 0,125 * \text{SampleRTT}$$

Amostras de RTTs estimados:

REDES DE
COMPUTADORES
E A INTERNET 5ª edição

Uma Abordagem Top-Down



Variações em SampleRTT são atenuadas em EstimatedRTT

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:

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

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

depois definir intervalo de timeout

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

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 sobre o serviço não confiável do IP
- ❑ segmentos em paralelo
- ❑ ACKs cumulativos
- ❑ TCP usa único temporizador de retransmissão (RFC 2988)
- ❑ retransmissões são disparadas por:
 - eventos de *timeout*
 - ACKs duplicados
- ❑ inicialmente, considera remetente TCP simplificado:
 - ignora ACKs duplicados
 - ignora controle de fluxo, controle de congestionamento

Eventos de remetente TCP:

dados recebidos da apl.:

- ❑ cria segmento com #seq
- ❑ #seq é número da cadeia de bytes do primeiro byte de dados no segmento
- ❑ inicia temporizador, se ainda não tiver iniciado (pense nele como para o segmento mais antigo sem ACK)
- ❑ intervalo de expiração:

TimeoutInterval

timeout:

- ❑ retransmite segmento que causou *timeout*
- ❑ reinicia temporizador

ACK recebido:

- ❑ Reconhecem-se segmentos sem ACK anteriores
 - atualiza o que sabidamente tem ACK
 - inicia temporizador se houver segmentos pendentes

RemetenteTCP (simplificado)

```
NextSeqNum = InitialSeqNum  
SendBase = InitialSeqNum
```

```
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(dados)
```

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

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

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

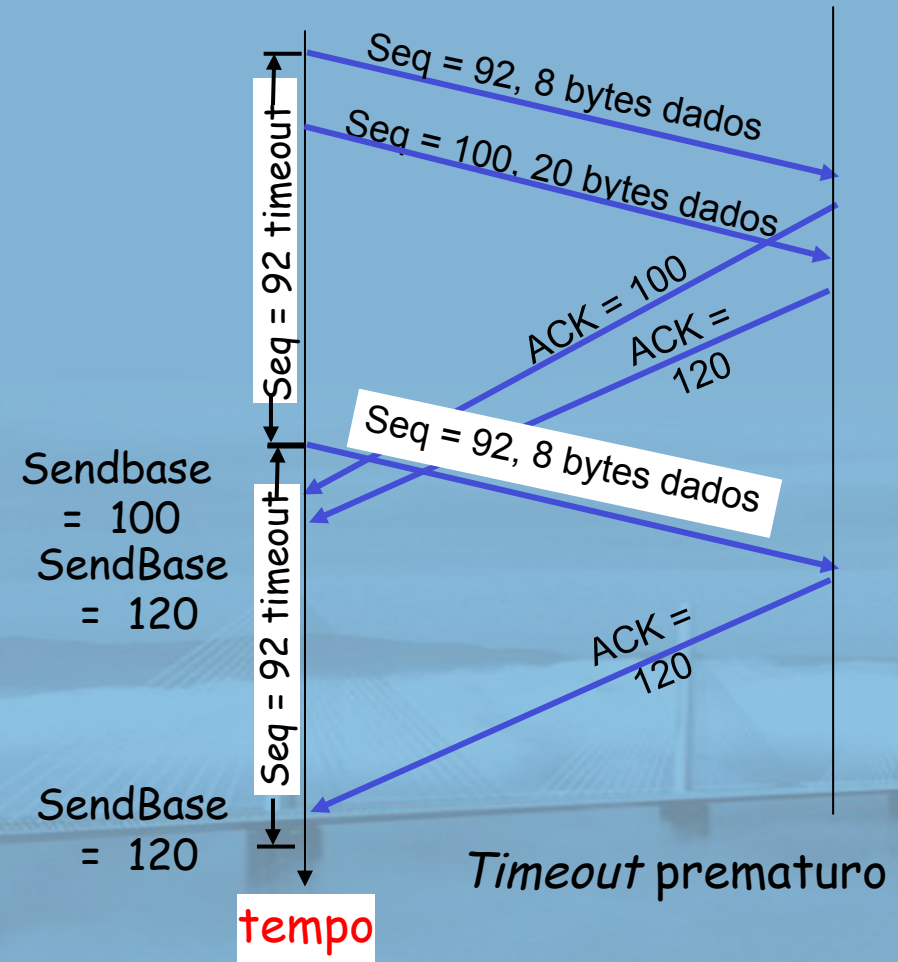
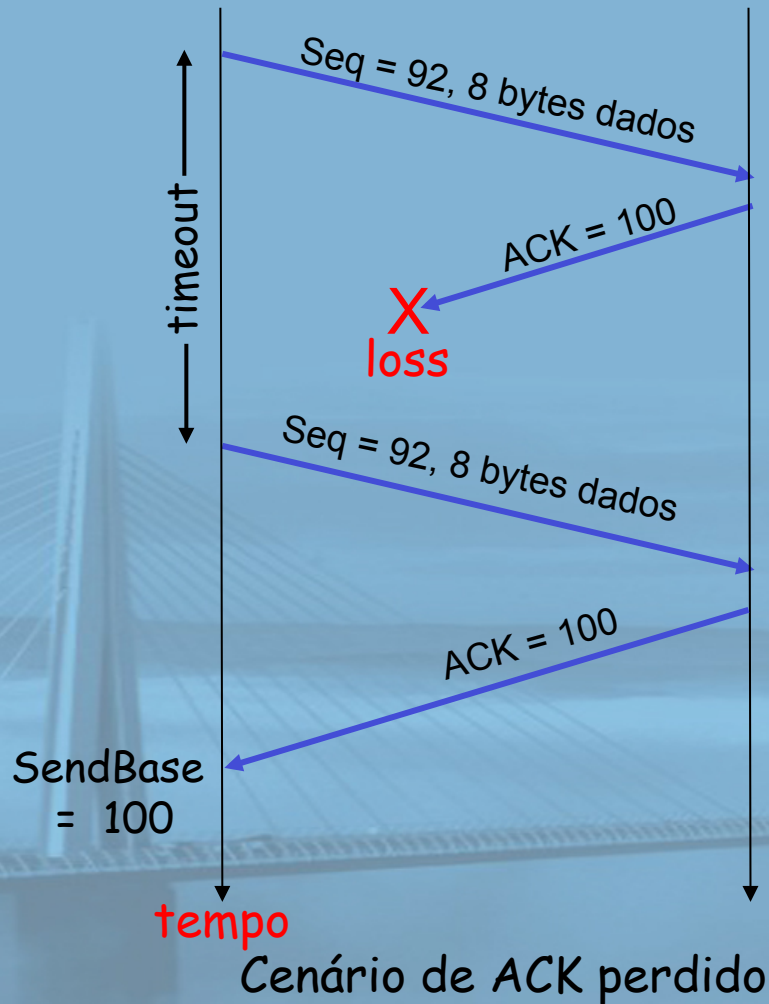
Comentário:

- SendBase-1: último byte cumulativo com ACK

TCP: cenários de retransmissão

REDES DE COMPUTADORES E A INTERNET

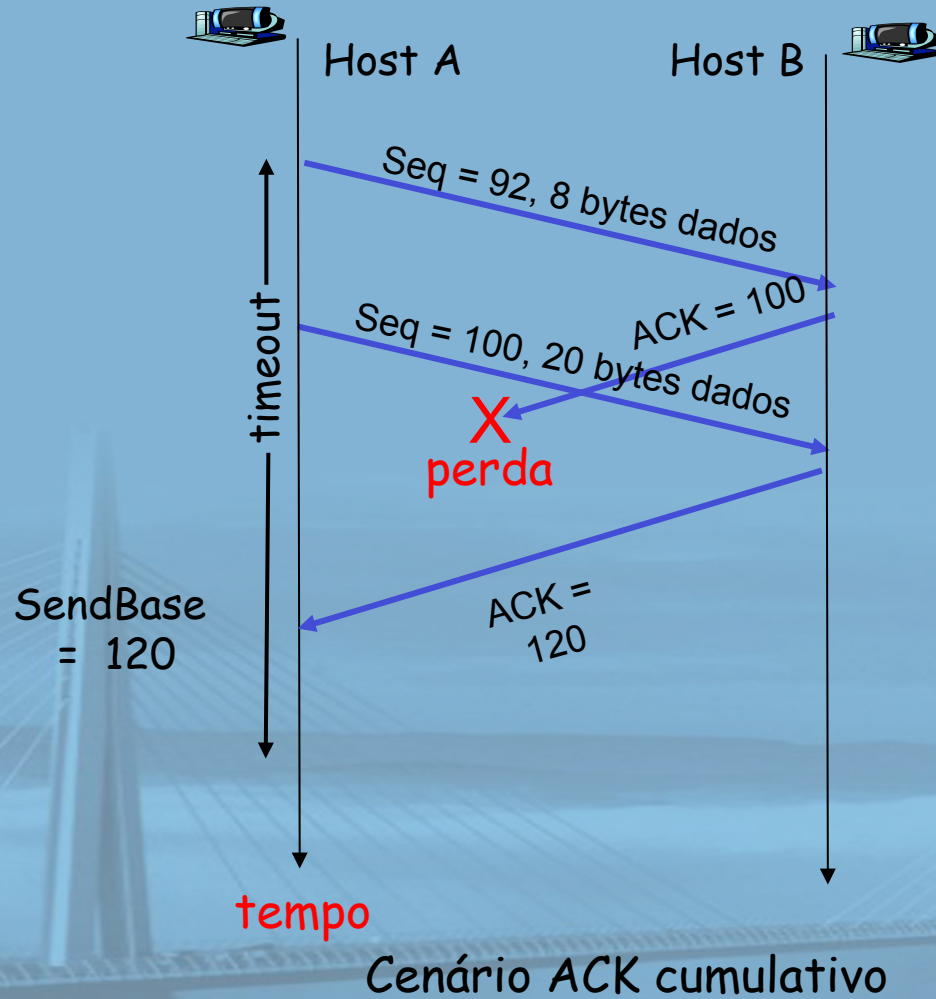
Uma Abordagem Top-Down



REDES DE COMPUTADORES E A INTERNET

5ª edição

Uma Abordagem Top-Down



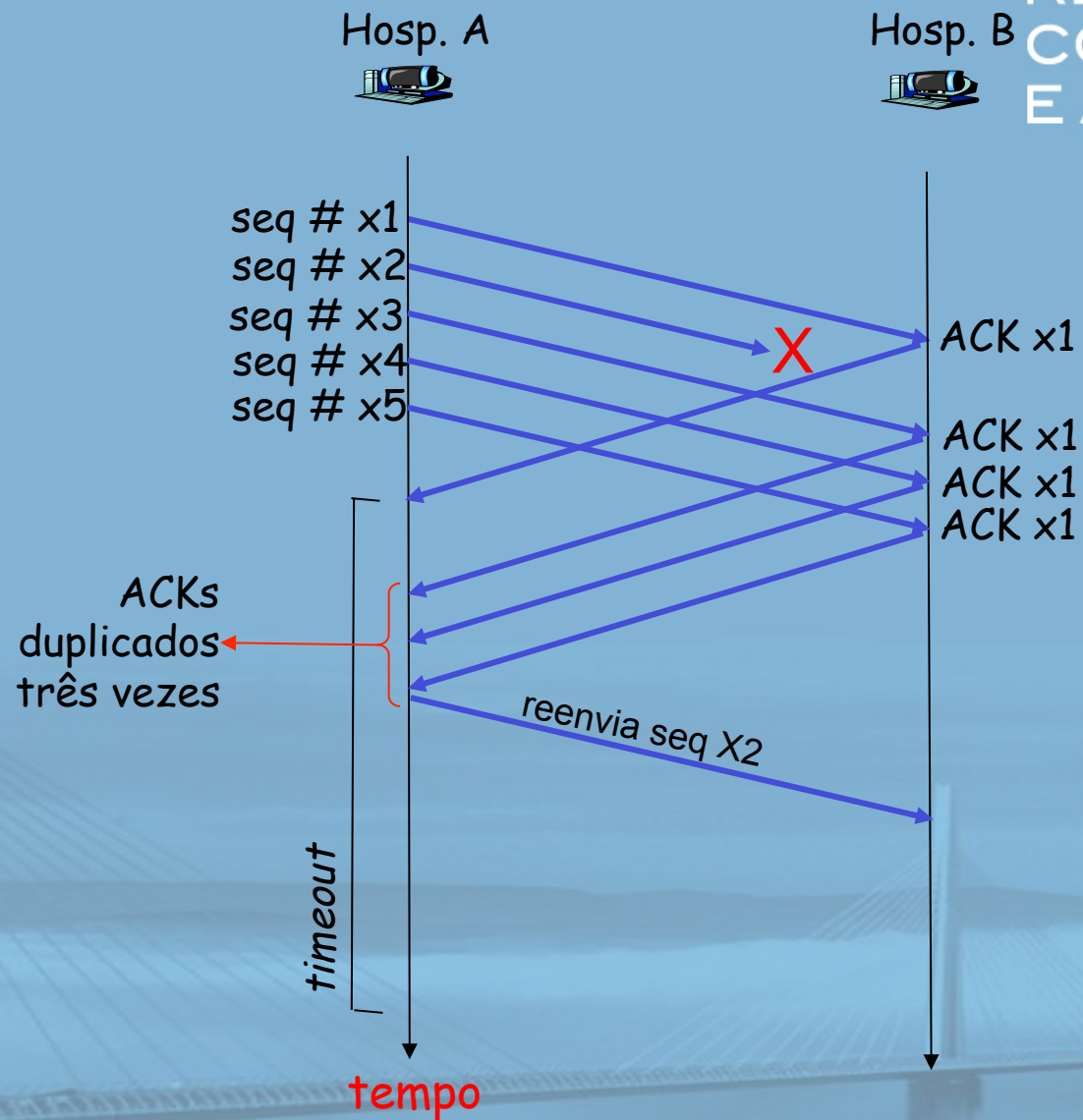
TCP: geração de ACK

[RFC 1122, RFC 2581]

Evento	Ação do TCP Destinatário
Chegada de segmento na ordem com número de sequência esperado. Todos os dados até o número de sequência esperado já reconhecidos.	ACK retardado. Espera de até 500 milissegundos pela chegada de um outro segmento na ordem. Se o segmento seguinte na ordem não chegar nesse intervalo, envia um ACK.
Chegada de segmento na ordem com número de sequência esperado. Um outro segmento na ordem esperando por transmissão de ACK.	Envio imediato de um único ACK cumulativo, reconhecendo ambos os segmentos na ordem.
Chegada de um segmento fora da ordem com número de sequência mais alto do que o esperado. Lacuna detectada.	Envio imediato de um ACK duplicado, indicando número de sequência do byte seguinte esperado (que é a extremidade mais baixa da lacuna).
Chegada de um segmento que preenche, parcial ou completamente, a lacuna nos dados recebidos.	Envio imediato de um ACK, contanto que o segmento comece na extremidade mais baixa da lacuna.

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



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
segmento já com ACK

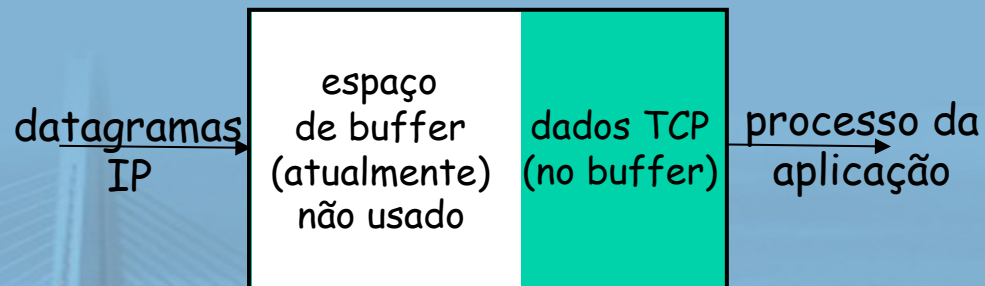
retransmissão rápida

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:



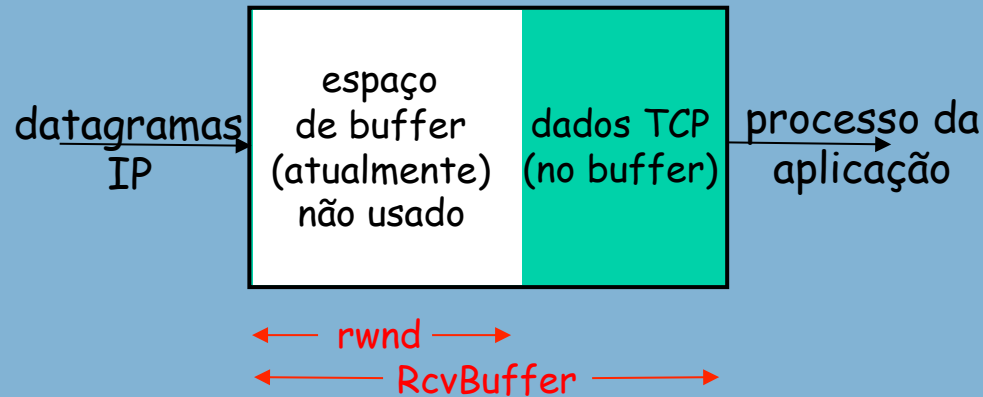
- ❑ processo da aplicação pode ser lento na leitura do buffer

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

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

$$rwnd = RcvBuffer - [LastByteRcvd - LastByteRead]$$

O que acontece no remetente?

❑ Monitora variáveis:

- LastByteSent
- LastByteAcked

❑ Certificar-se que:

- $\text{LastByteSent} - \text{LastByteAcked} \leq \text{rwnd}$

❑ Problema!!!

- O que acontece se host destino informar $\text{rwnd} = 0$?
(indicando buffer cheio)

❑ Ver applet (controle de fluxo)

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 de 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: host cliente envia segmento SYN do TCP ao servidor

- especifica # seq. inicial
- sem dados

etapa 2: host 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

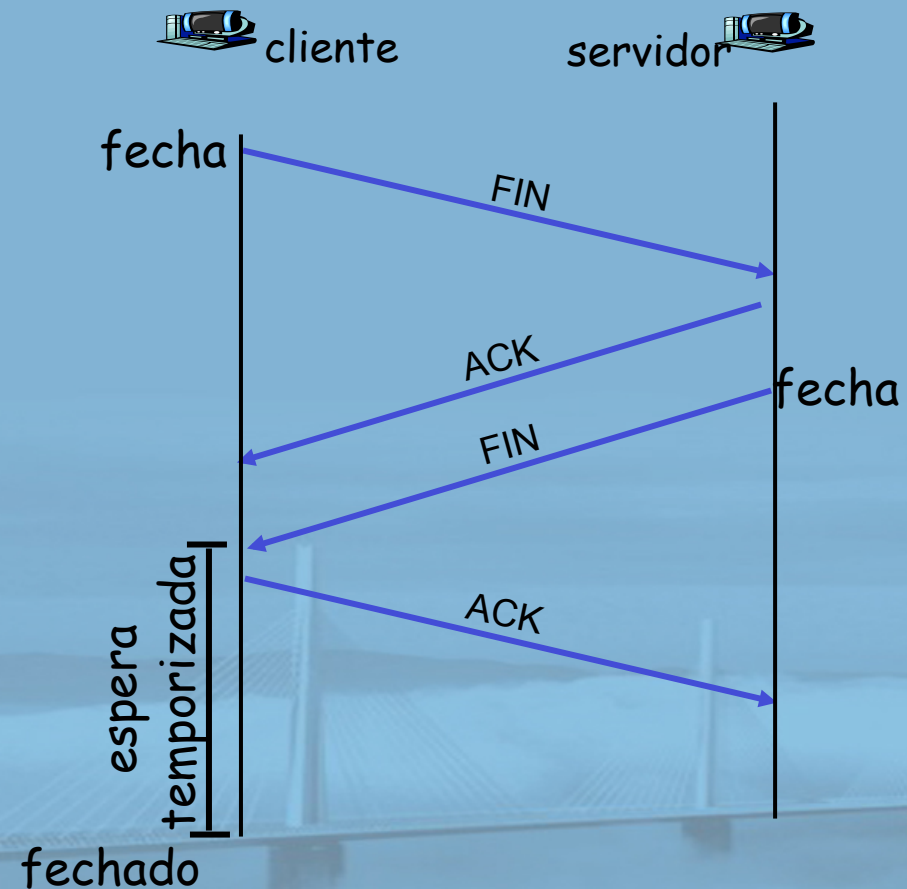
fechando uma conexão:

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.

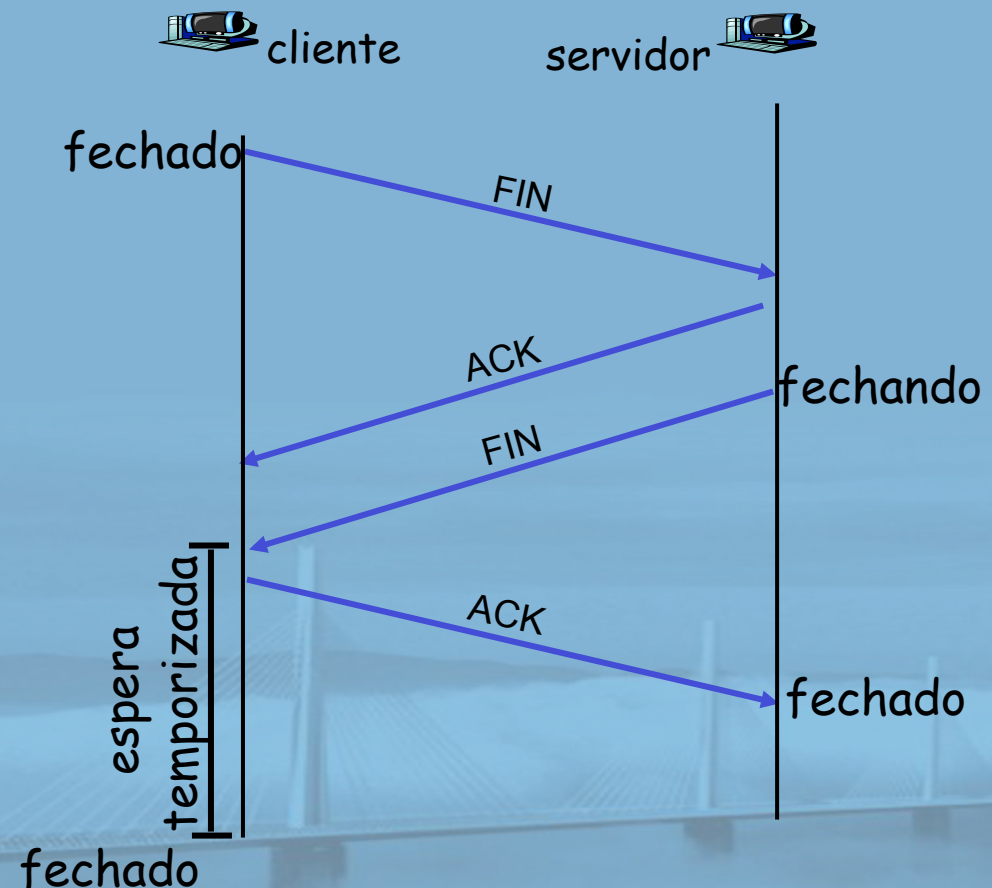


etapa 3: cliente recebe
FIN, responde com ACK

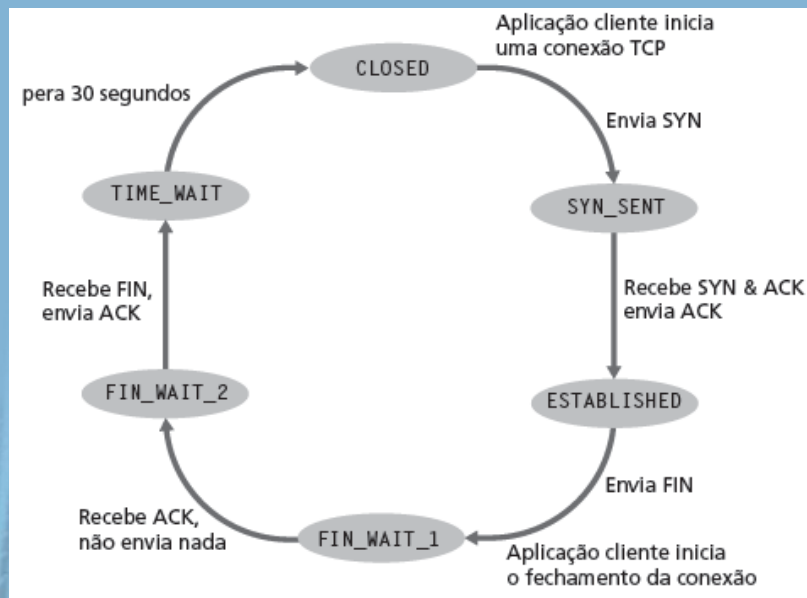
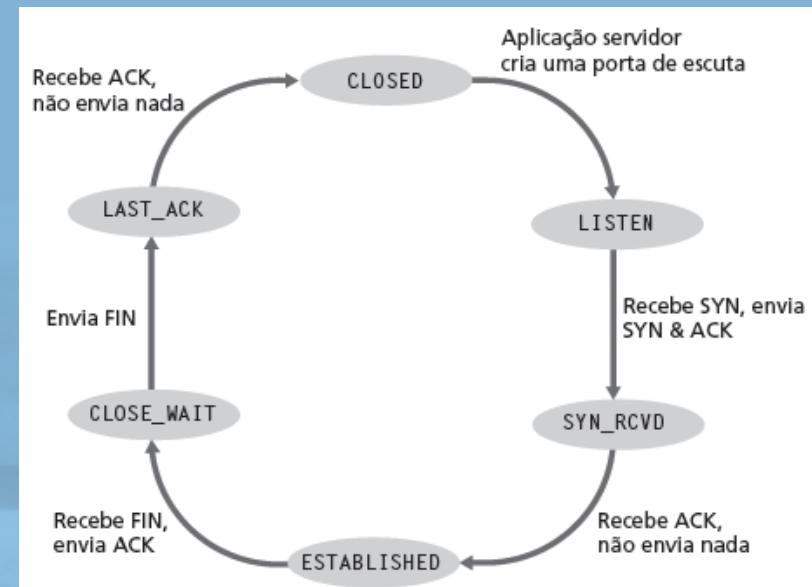
- entra em "espera temporizada" - responderá com ACK aos FINs recebidos

etapa 4: servidor recebe
ACK - conexão fechada

Nota: Com pequena modificação, pode tratar de FINs simultâneos.



ciclo de vida do servidor TCP



ciclo de vida do cliente 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

Congestionamento:

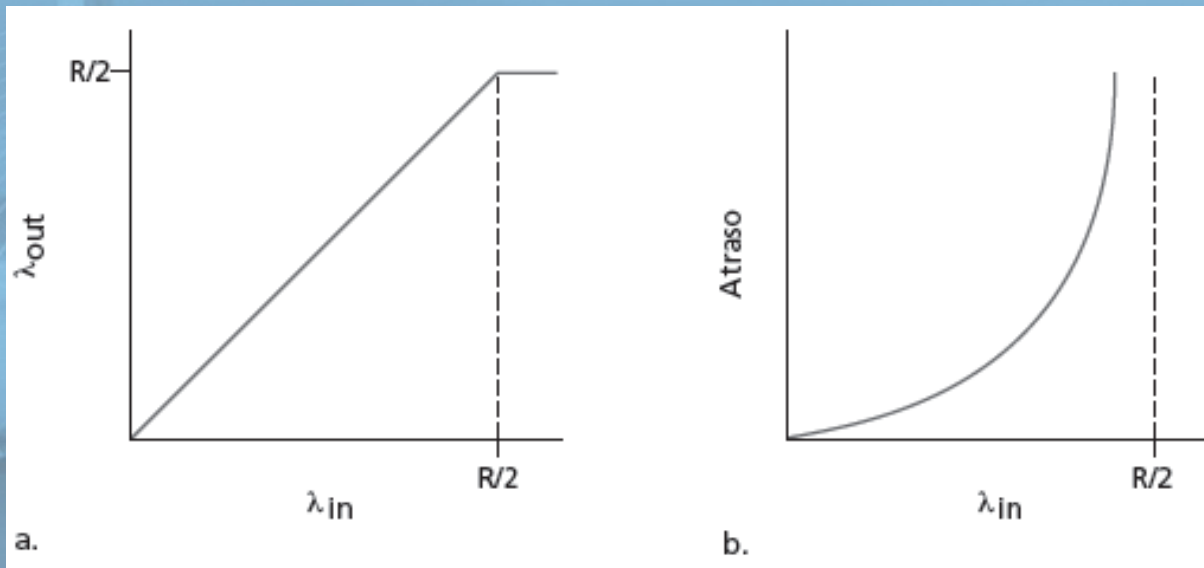
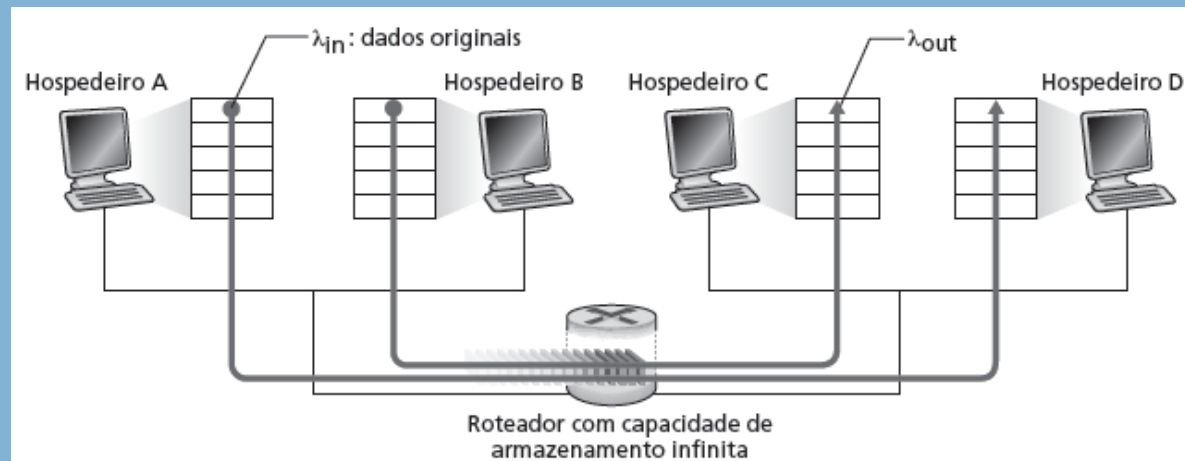
- ❑ informalmente: “muitas fontes enviando muitos dados muito rápido para a *rede* tratar”
- ❑ diferente de controle de fluxo!
- ❑ manifestações:
 - pacotes perdidos (estouro de buffer nos roteadores)
 - longos atrasos (enfileiramento nos buffers do roteador)
- ❑ um dos maiores problemas da rede!

Causas/custos do congestionamento: cenário 1

REDES DE
COMPUTADORES
E A INTERNET 5ª edição

Uma Abordagem Top-Down

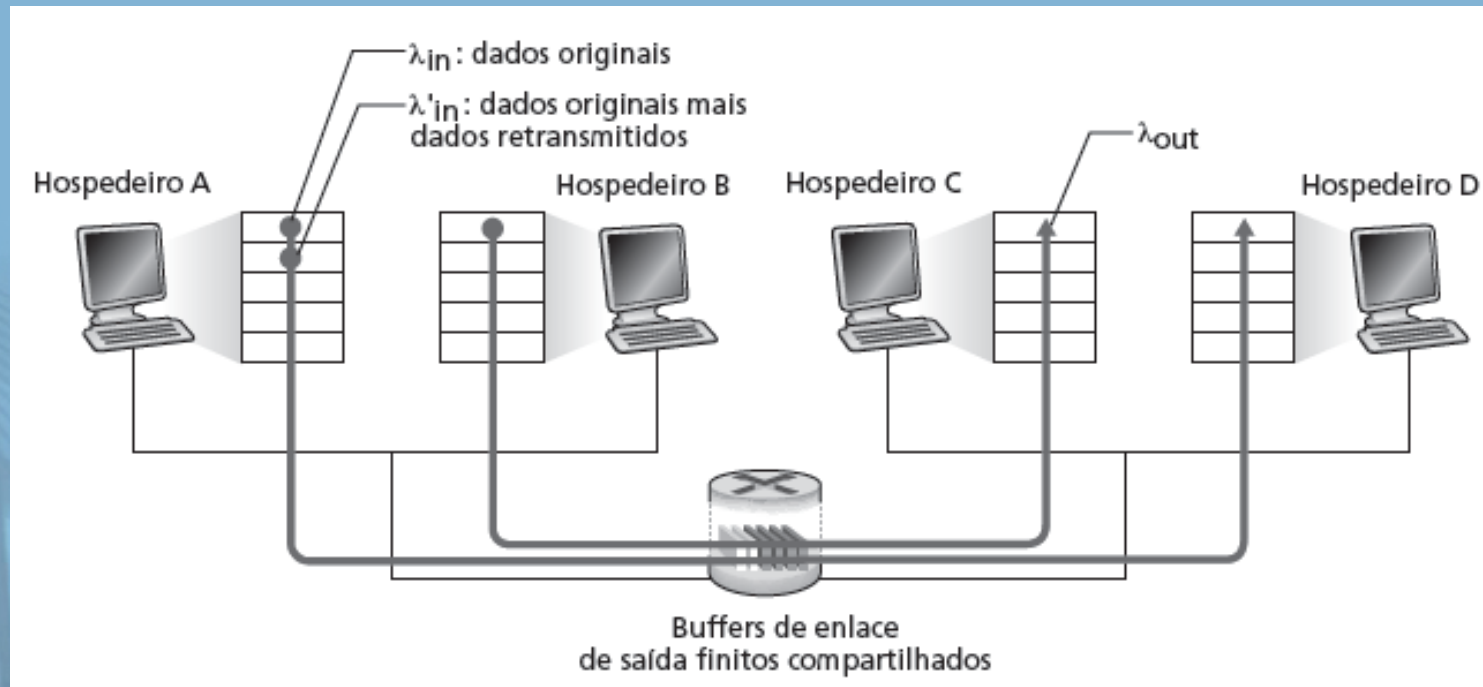
- ❑ dois remetentes, dois destinatários
- ❑ um roteador, infinitos buffers
- ❑ sem retransmissão



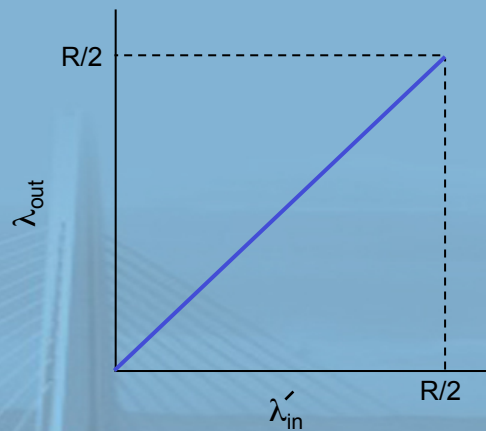
- ❑ grandes atrasos quando congestionado
- ❑ vazão máxima alcançável

Causas/custos do congestionamento: cenário 2

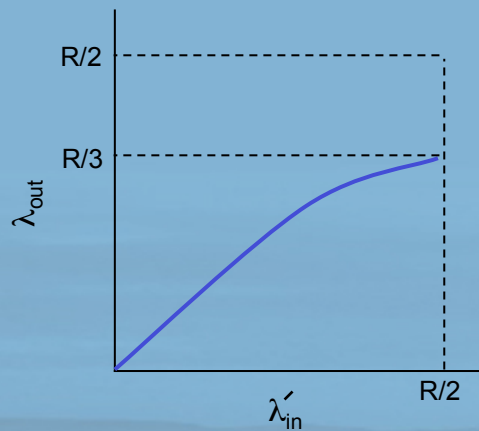
- ❑ um roteador, buffers *finitos*
- ❑ retransmissão do pacote perdido pelo remetente



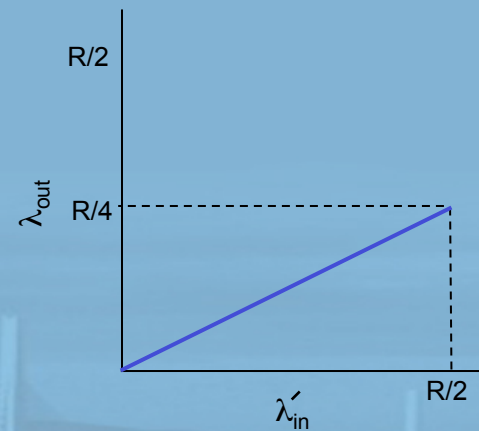
- sempre: $\lambda_{in} = \lambda_{out}$ (vazão)
- retransmissão "perfeita" apenas quando há perda: $\lambda'_{in} > \lambda_{out}$
- retransmissão do pacote com timeout prematuro torna λ'_{in} maior (que o caso perfeito) para o mesmo λ_{out}



a.



b.



c.

"custos" do congestionamento:

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

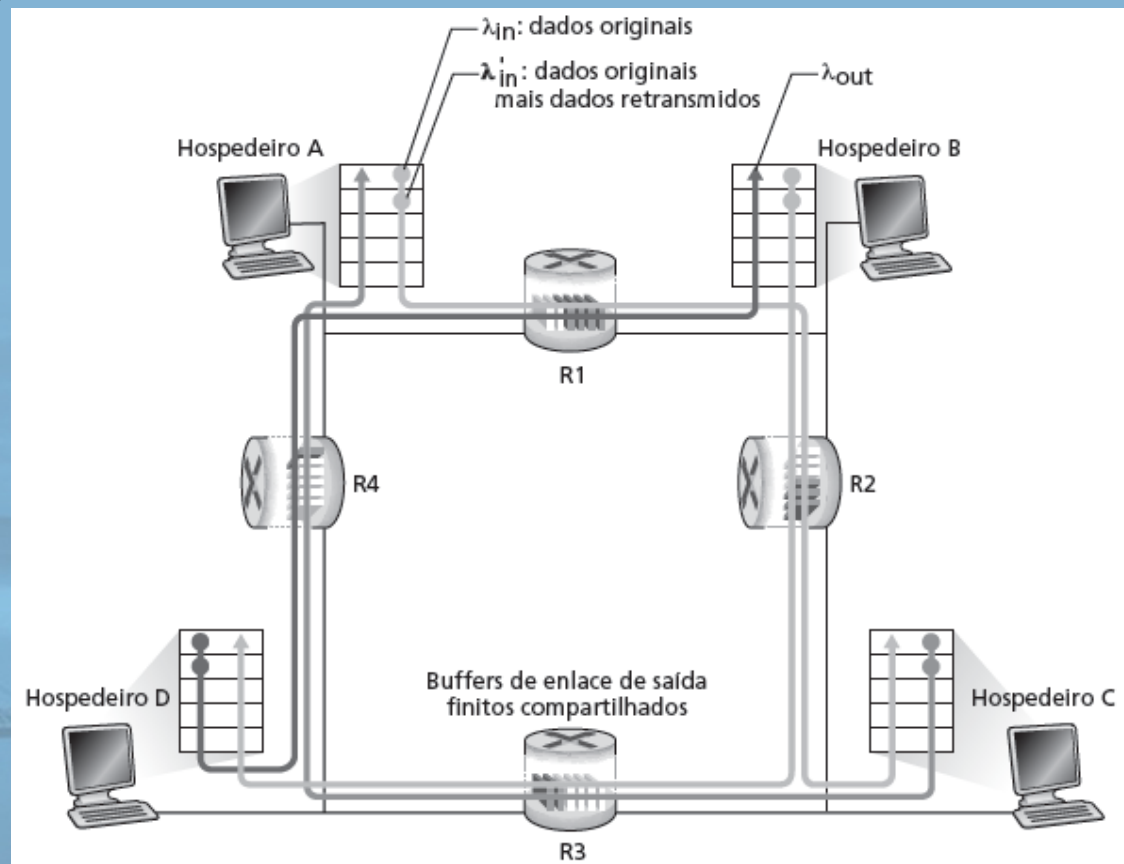
Causas/custos do congestionamento: cenário 3

REDES DE
COMPUTADORES
E A INTERNET 5ª edição

Uma Abordagem Top-Down

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

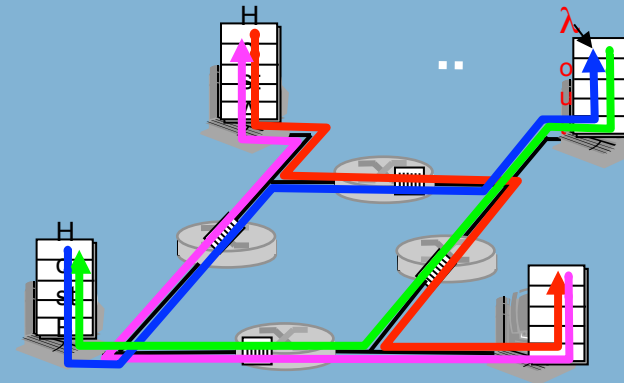
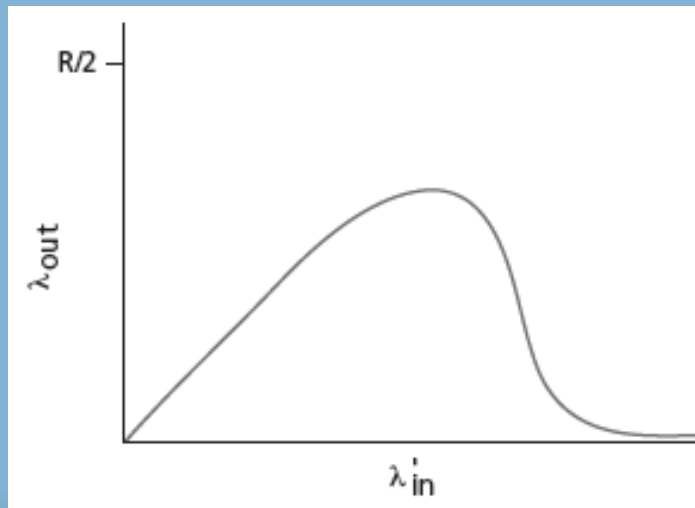
P: O que acontece quando λ_{in} e λ'_{in} aumentam?



REDES DE COMPUTADORES E A INTERNET

5ª edição

Uma Abordagem Top-Down



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 (SNA, DECbit, TCP/IP ECN, ATM)
 - taxa explícita que o remetente deve enviar no enlace de saída

Estudo de caso: controle de congestionamento ATM ABR

ABR: taxa de bit disponível:

- ❑ "serviço elástico"
- ❑ se caminho do remetente "sobrecarregado":
 - remetente deve usar largura de banda disponível
- ❑ se caminho do remetente congestionado:
 - remetente sufocado à taxa mínima garantida

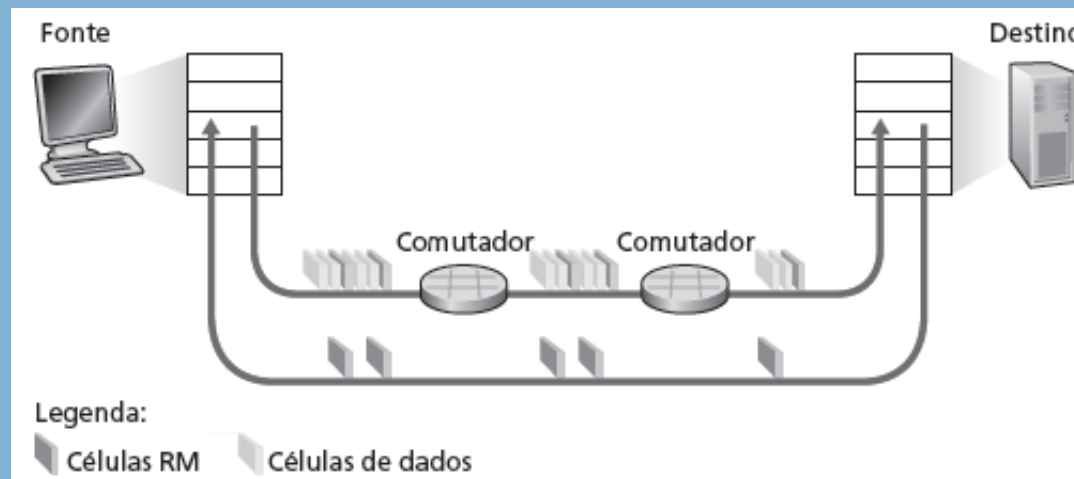
células RM (gerenciamento de recursos) :

- ❑ enviadas pelo remetente, intercaladas com células de dados
- ❑ bits na célula RM definida por comutadores ("*assistido pela rede*")
 - bit **NI**: sem aumento na taxa (congestionamento leve)
 - bit **CI**: indicação de congestionamento
- ❑ células RM retornadas ao remetente pelo destinatário, com bits intactos

REDES DE COMPUTADORES E A INTERNET

5ª edição

Uma Abordagem Top-Down



- ❑ campo ER (explicit rate) de 2 bytes na célula RM
 - comutador congestionado pode reduzir valor de ER na célula
 - taxa de envio do remetente é taxa máxima admissível no caminho
- ❑ bit EFCI nas células de dados: define como 1 no comutador congestionado
 - se a célula de dados anterior à célula RM tiver EFCI definido, remetente define bit CI na célula RM retornada

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

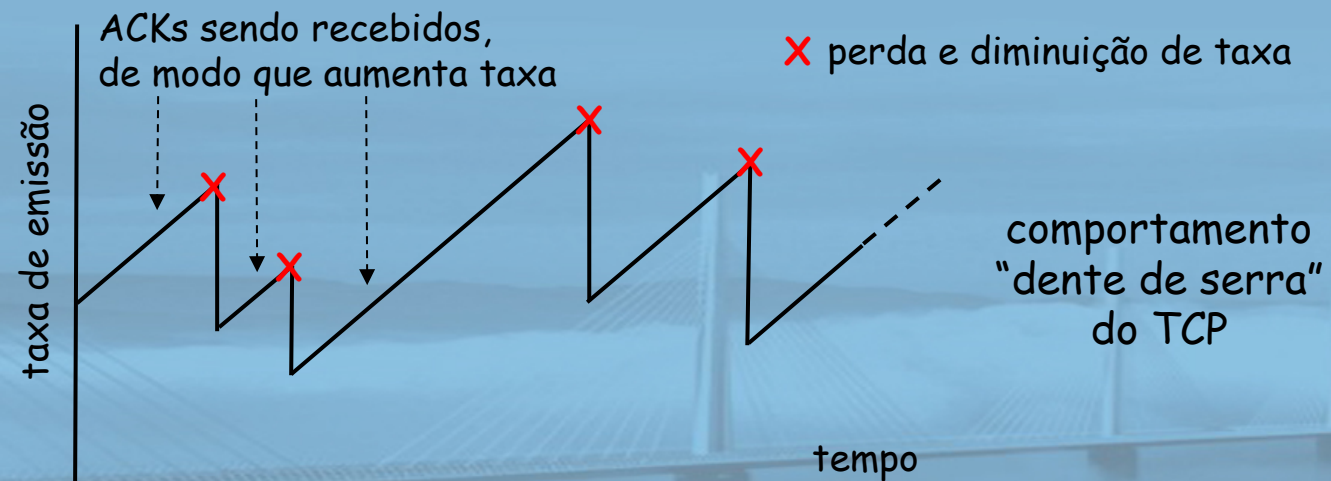
Controle de congestionamento

TCP: busca por largura de banda de banda

REDES DE
COMPUTADORES
E A INTERNET 5ª edição

Uma Abordagem Top-Down

- “procura por largura de banda”: aumenta taxa de transmissão no recebimento do ACK até por fim ocorrer perda; depois diminui taxa de transmissão
 - continua a aumentar no ACK, diminui na perda (pois largura de banda disponível está mudando, dependendo de outras conexões na rede)



- P: Com que velocidade aumentar/diminuir?
 - detalhes a seguir

Controle de congestionamento

TCP: detalhes

- ❑ remetente limita taxa limitando número de bytes sem ACK "na pipeline":

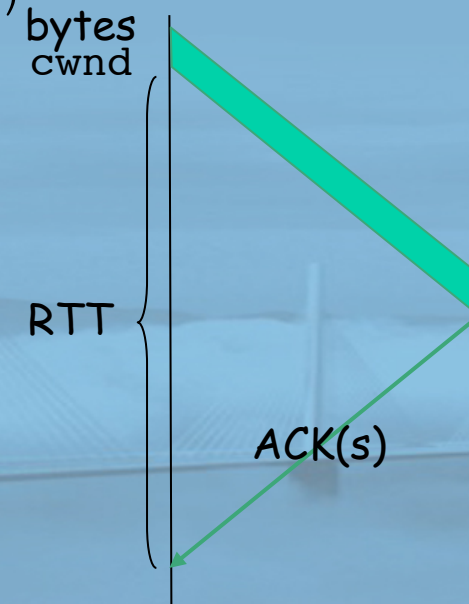
$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$

- cwnd: difere de rwnd
- remetente limitado por $\min(\text{cwnd}, \text{rwnd})$

- ❑ aproximadamente,

$$\text{taxa} = \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/seg}$$

- ❑ cwnd é dinâmico, função do congestionamento de rede percebido



Controle de congestionamento

TCP: mais detalhes

evento de perda de segmento: reduzindo cwnd

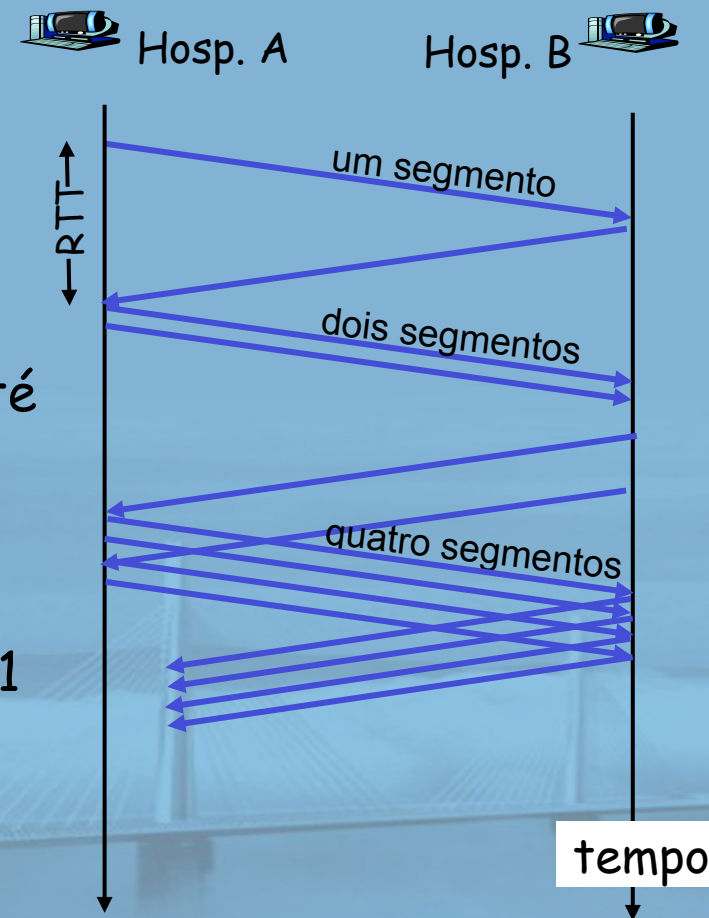
- ❑ *timeout*: sem resposta do destinatário
 - corta cwnd para 1
- ❑ 3 ACKs duplicados: pelo menos alguns segmentos passando (lembre-se da retransmissão rápida)
 - corta cwnd pela metade, menos agressivo do que no *timeout*

ACK recebido: aumenta cwnd

- ❑ fase de partida lenta:
 - aumento exponencialmente rápido (apesar do nome) no início da conexão, ou após o *timeout*
- ❑ prevenção de congestionamento:
 - aumento linear

Partida lenta do TCP

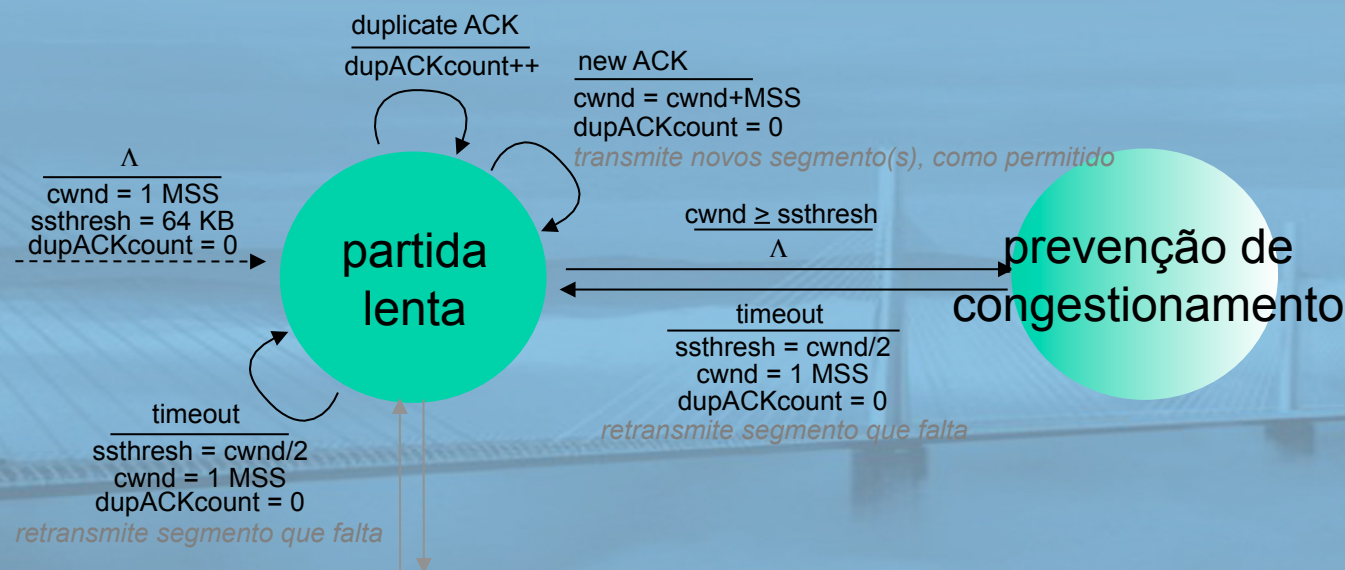
- ❑ quando conexão começa:
 - $cwnd = 1 \text{ MSS}$
- ❑ largura de banda disponível pode ser $\gg \text{MSS}/\text{RTT}$
 - desejável subir rapidamente para taxa respeitável
- ❑ aumenta taxa exponencialmente até o primeiro evento de perda ou quando o patamar é alcançado
 - $cwnd$ duplo a cada RTT
 - feito incrementando $cwnd$ por 1 para cada ACK recebido



Transição dentro/fora da partida rápida

ssthresh: patamar de **cwnd** mantido pelo TCP

- um evento de perda: define **ssthresh** como **cwnd/2**
 - lembre-se (metade) da taxa TCP quando ocorreu perda de congestionamento
- quando transição de $\text{cwnd} \geq \text{ssthresh}$: da partida lenta para fase de prevenção de congestionamento



TCP: prevenção de congestionamento

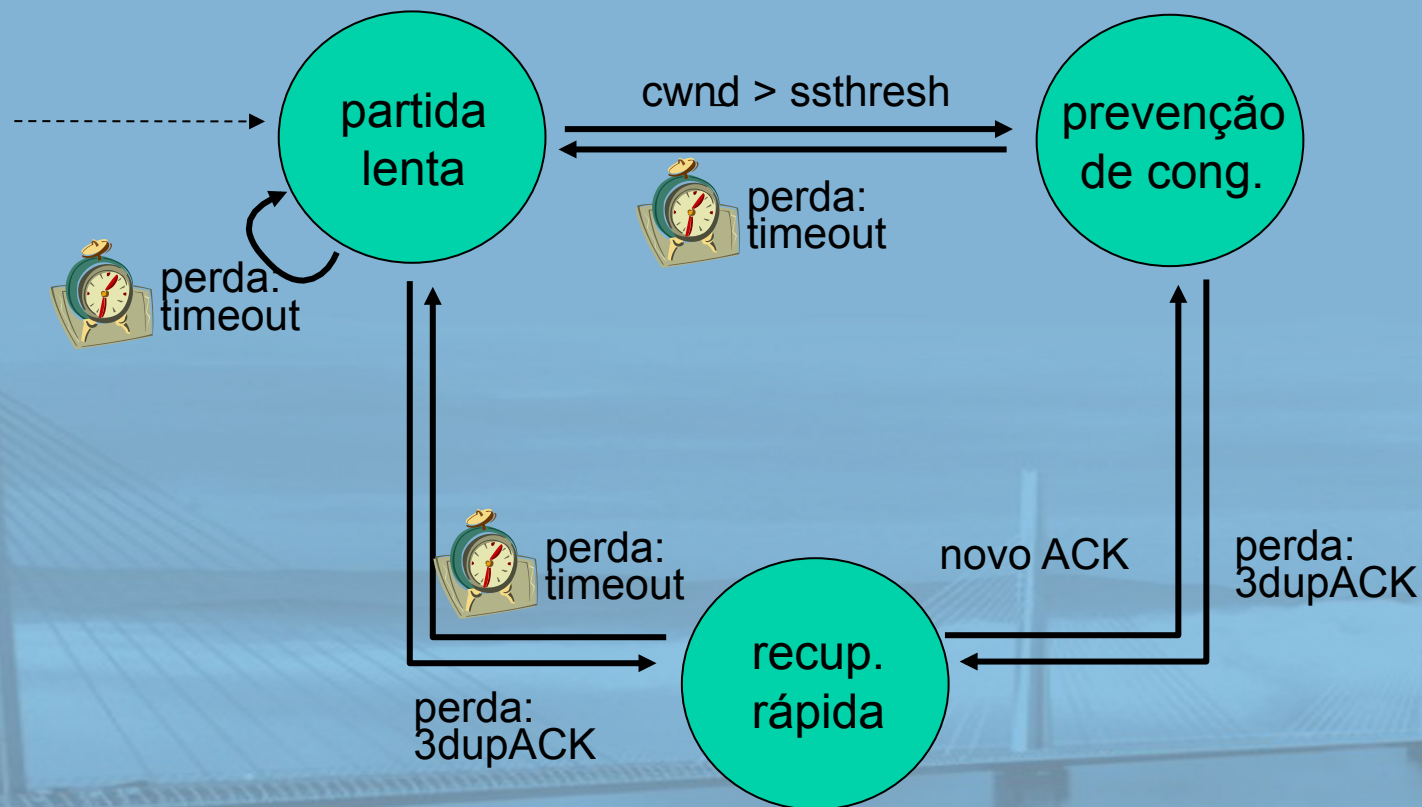
- quando $cwnd > ssthresh$
cresce $cwnd$ de forma linear
 - aumenta $cwnd$ em 1 MSS por RTT
 - aborda possível congestionamento mais lento que na partida lenta
 - implementação: $cwnd = cwnd + MSS/cwnd$ para cada ACK recebido

AIMD

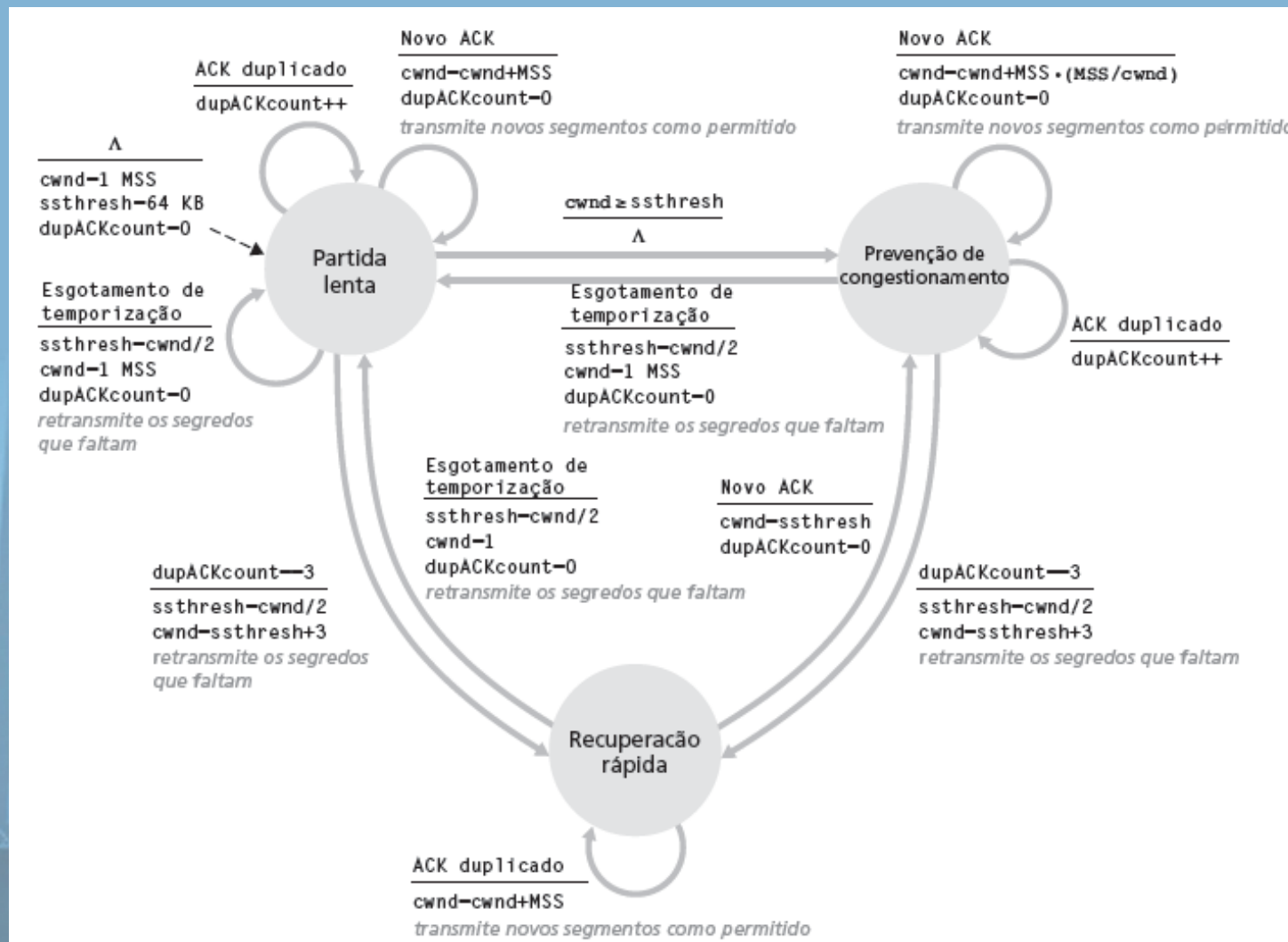
- **ACKs:** aumenta $cwnd$ em 1 MSS por RTT: aumento aditivo
- **perda:** corta $cwnd$ ao meio (perda sem *timeout* detectado): diminuição multiplicativa

AIMD: Additive Increase
Multiplicative Decrease

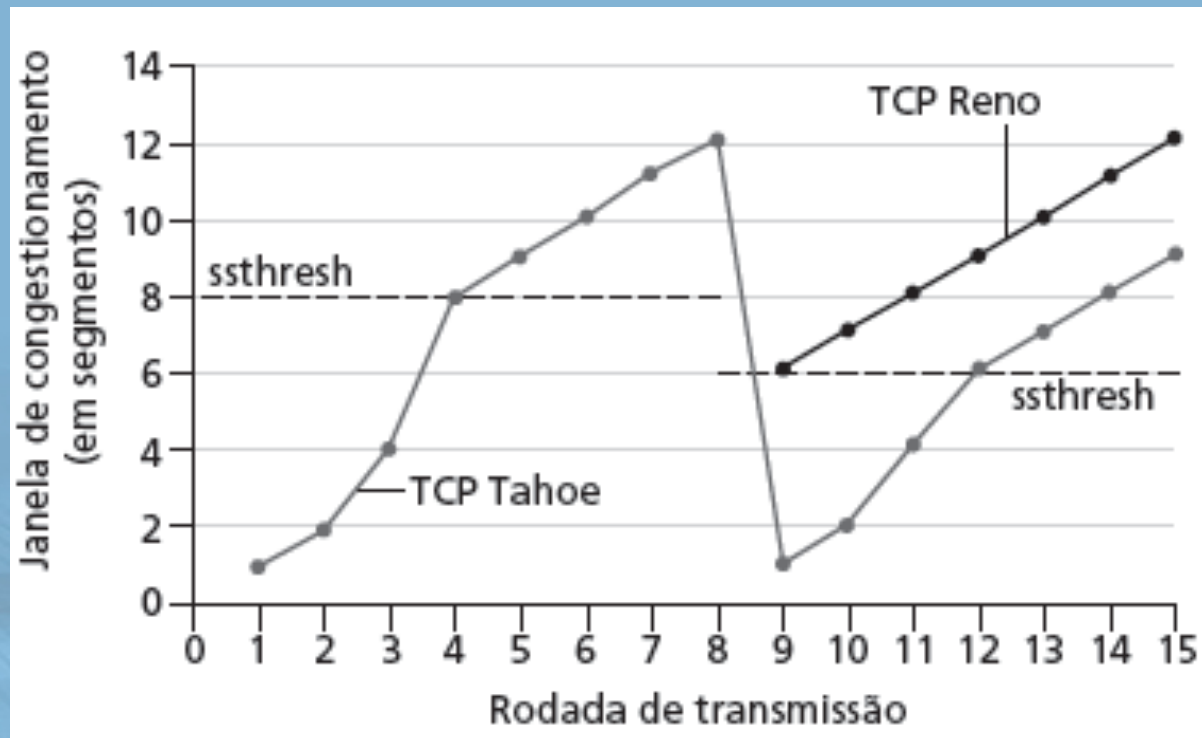
FSM do controle de congestionamento TCP: visão geral



FSM do controle de congestionamento TCP: detalhes



Tipos populares de TCP



Resumo: controle de congestionamento TCP

- ❑ quando $cwnd < ssthresh$, remetente na fase de **partida lenta**, janela cresce exponencialmente.
- ❑ quando $cwnd \geq ssthresh$, remetente está na fase de **prevenção de congestionamento**, janela cresce linearmente.
- ❑ quando ocorre o **ACK duplicado triplo**, $ssthresh$ definido como $cwnd/2$, $cwnd$ definido como $\sim ssthresh$
- ❑ quando ocorre o **timeout**, $ssthresh$ definido como $cwnd/2$, $cwnd$ definido como 1 MSS.

Vazão do TCP

- ❑ P: Qual é a vazão média do TCP como função do tamanho da janela, RTT?
 - ignorando partida lenta
- ❑ seja W o tamanho da janela quando ocorre a perda
 - quando janela é W , a vazão é W/RTT
 - logo após perda, janela cai para $W/2$, vazão para $W/2RTT$.
 - após a vazão: $0,75 W/RTT$

Futuros do TCP: TCP sobre pipes "longos, gordos"

- ❑ exemplo: segmentos de 1500 bytes, RTT de 100 ms, deseja vazão de 10 Gbps
- ❑ exige tamanho de janela $W = 83.333$ segmentos no ar

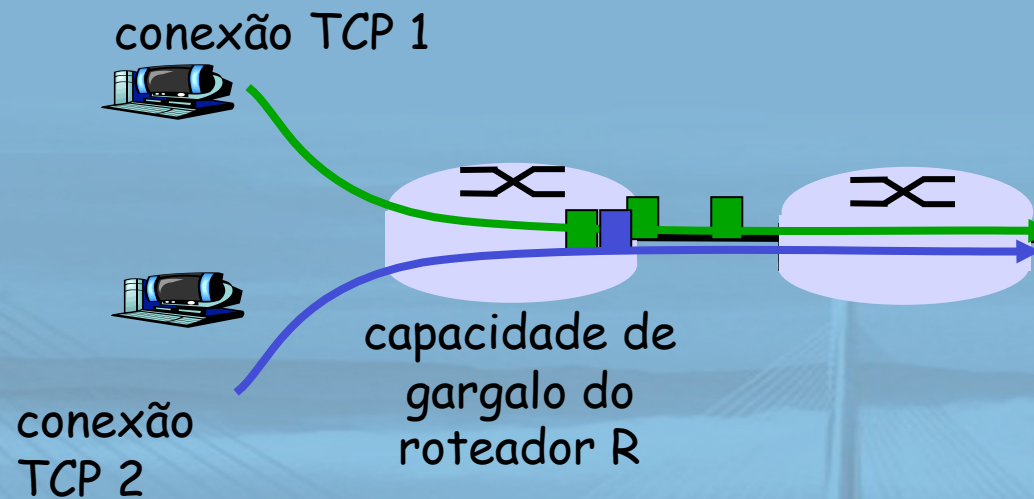
- ❑ vazão em termos da taxa de perda:

$$\frac{1,22 \cdot MSS}{RTT \sqrt{L}}$$

- ❑ $\rightarrow L = 2 \cdot 10^{-10}$ *Uau!*
- ❑ novas versões do TCP para alta velocidade

Equidade do TCP

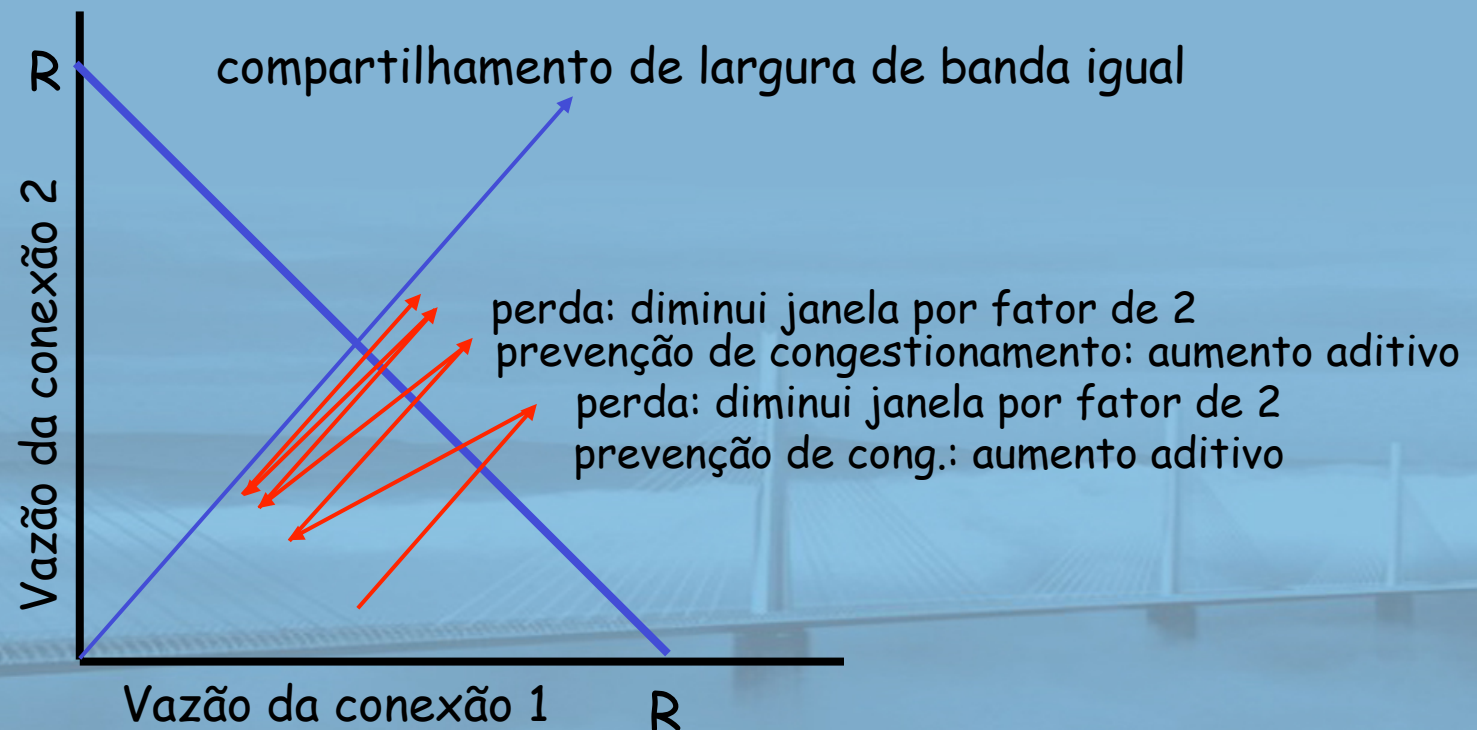
objetivo da equidade: se K sessões TCP
compartilharem o mesmo enlace de gargalo da
largura de banda R , cada uma deve ter uma taxa
média de R/K



Por que o TCP é justo?

duas sessões concorrentes:

- aumento aditivo dá inclinação 1, pois vazão aumenta
- diminuição multiplicativa diminui vazão proporcionalmente



Equidade (mais)

equidade e UDP

- ❑ aplicações de multimídia normalmente não usam TCP
 - não desejam que a taxa seja sufocada pelo controle de congestionamento
- ❑ em vez disso, use UDP:
 - envia áudio/vídeo em taxa constante, tolera perdas de pacotes

equidade e conexões TCP paralelas

- ❑ nada impede que a aplicação abra conexões paralelas entre 2 hospedeiros.
- ❑ navegadores Web fazem isso
- ❑ exemplo: enlace de taxa R admitindo 9 conexões;
 - nova aplicação solicita 1 TCP, recebe taxa $R/10$
 - nova aplicação solicita 11 TCPs, recebe $R/2$!

Capítulo 3: Resumo

- ❑ princípios por trás dos serviços da camada de transporte:
 - multiplexação, demultiplexação
 - transferência de dados confiável
 - controle de fluxo
 - controle de congestionamento
- ❑ instância e implementação na Internet
 - UDP
 - TCP

Em seguida:

- ❑ saindo da "borda" da rede (camada de transportes da aplicação)
- ❑ no "núcleo" da rede