Camada de Transporte (cont.)

REDES DE COMPUTADORES E A INTERNET 5ª edição
Uma Abordagem Top-Down

- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não orientado para conexão: UDP
- Princípios da transferência confiável de dados
- Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- Princípios de controle de congestionamento
- Congestionamento no TCP

Desempenho do rdt3.0

- rdt3.0 funciona, mas com desempenho ruim
- ex.: enlace 1 Gbps, 15 ms atraso propriedade, pacote 8000 bits:

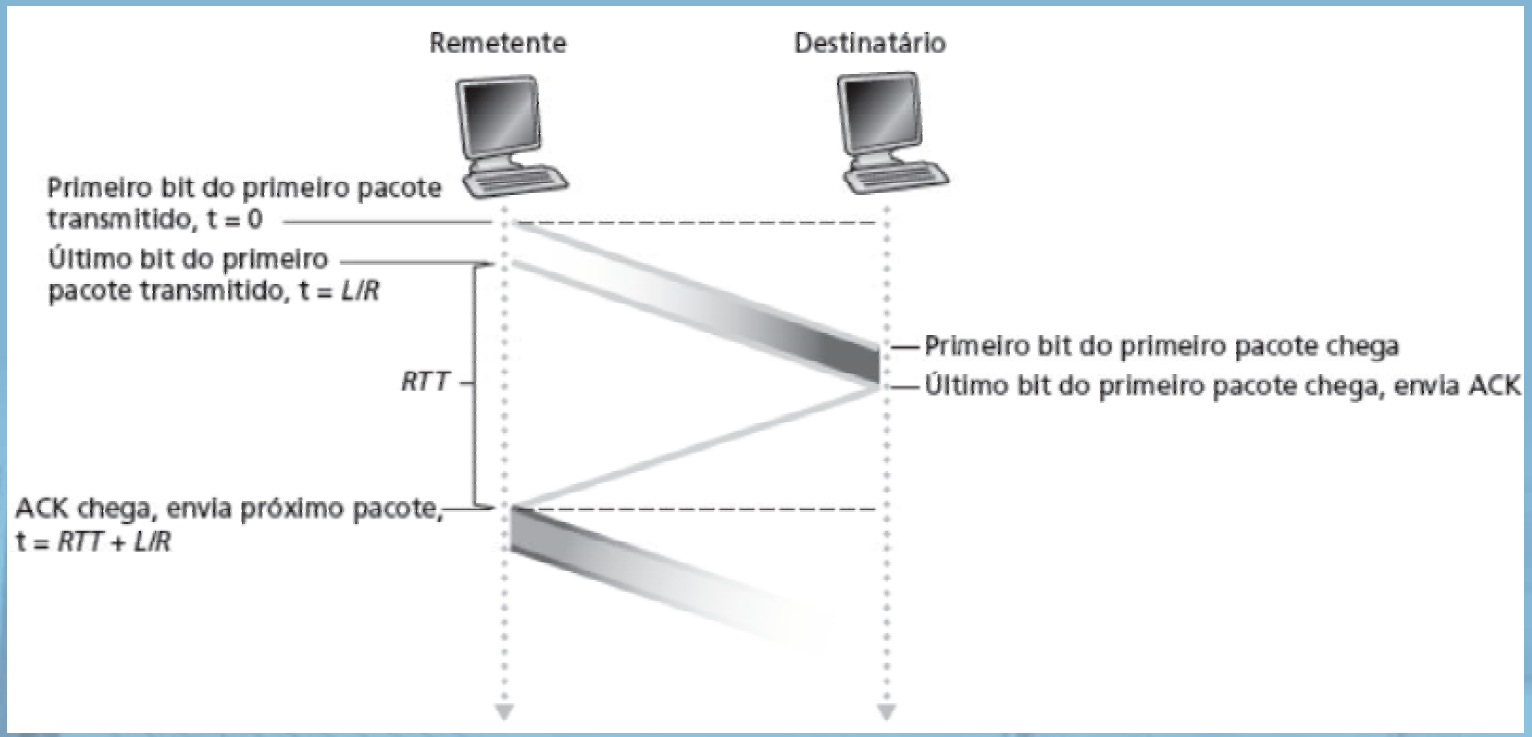
$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits/packet}}{10^9 \text{ bits/sec}} = 8 \text{ microseconds}$$

- 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 vazão em enlace de 1 Gbps
- protocolo de rede limita uso de recursos físicos!

rdt3.0: operação pare e espere

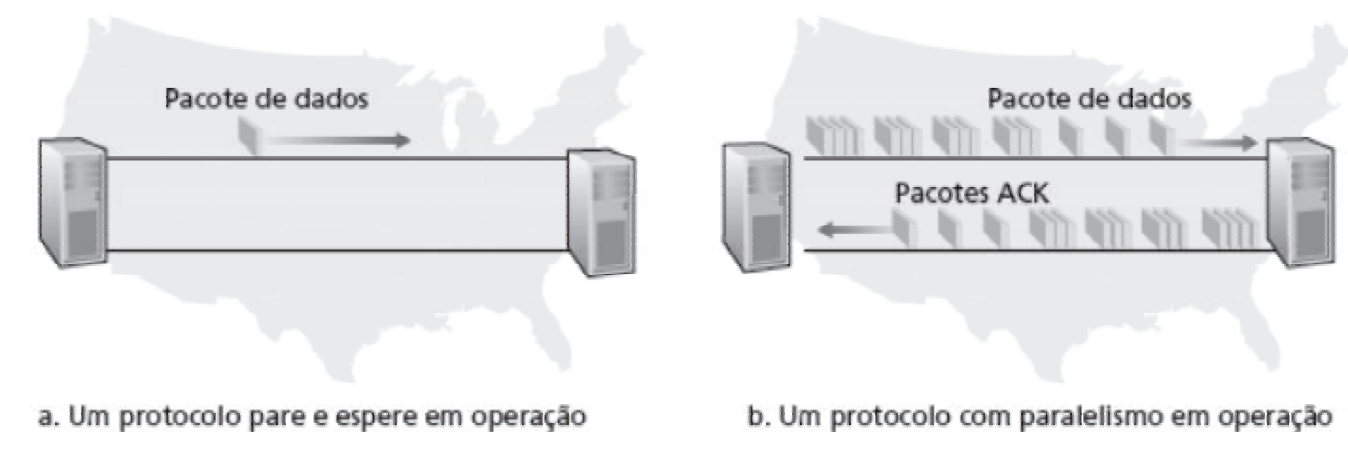


$$U_{remet} = \frac{L / R}{RTT + L / R} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

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

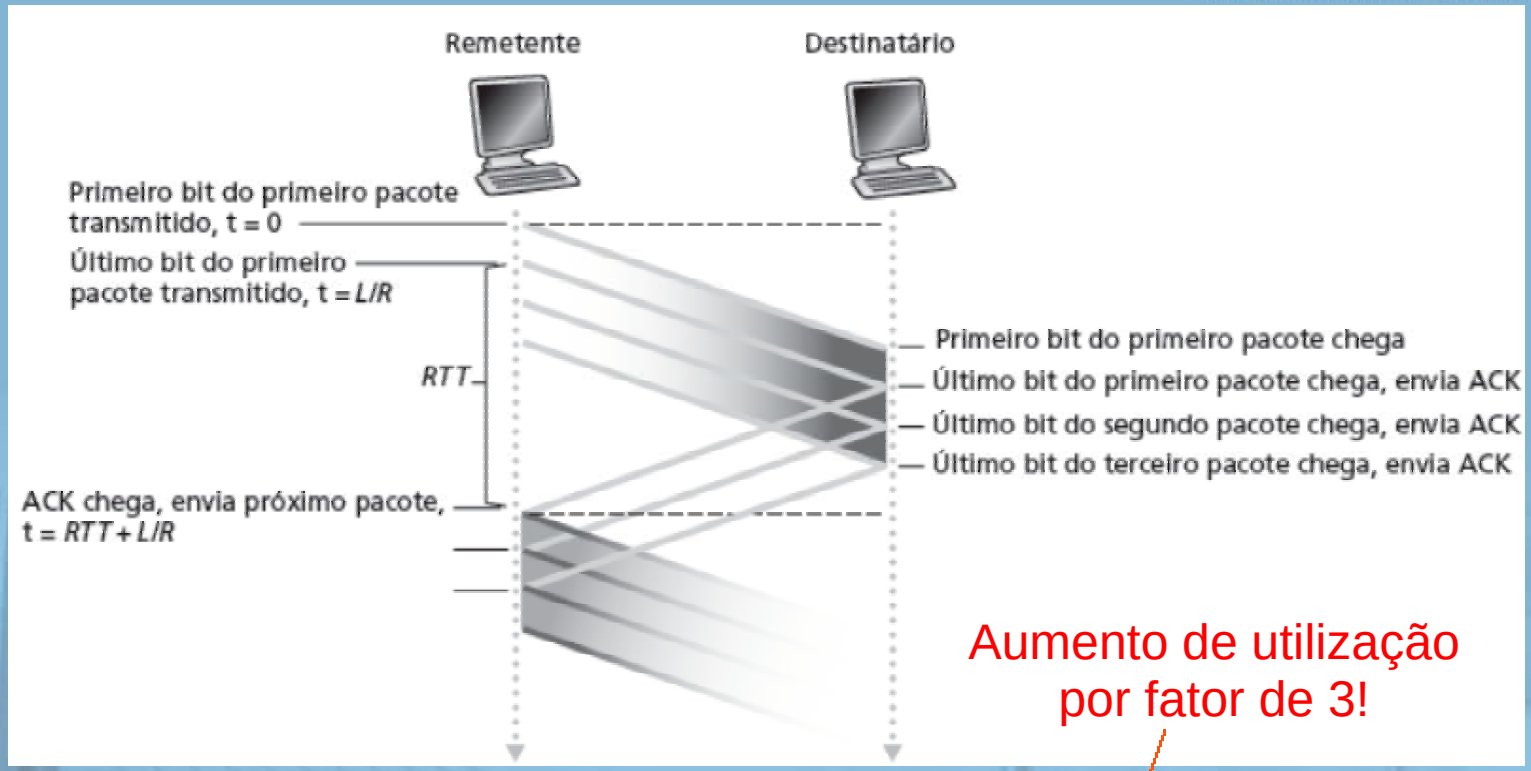


a. Um protocolo pare e espere em operação

b. Um protocolo com paralelismo em operação

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

Paralelismo: utilização aumentada



Aumento de utilização por fator de 3!

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

Protocolos com paralelismo

Go-back-N: visão geral

- *remetente:* até N pacotes não reconhecidos na pipeline
- *destinatário:* só 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

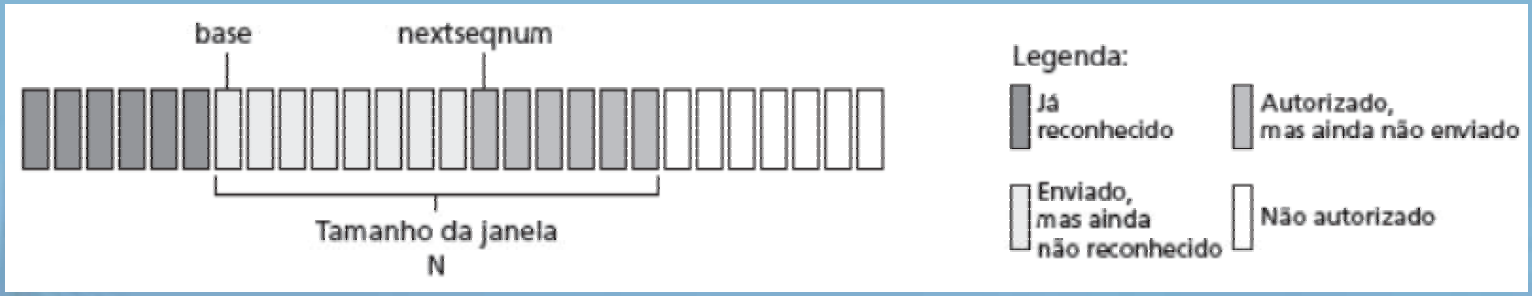
Repetição seletiva: visão geral

- *remetente:* até 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

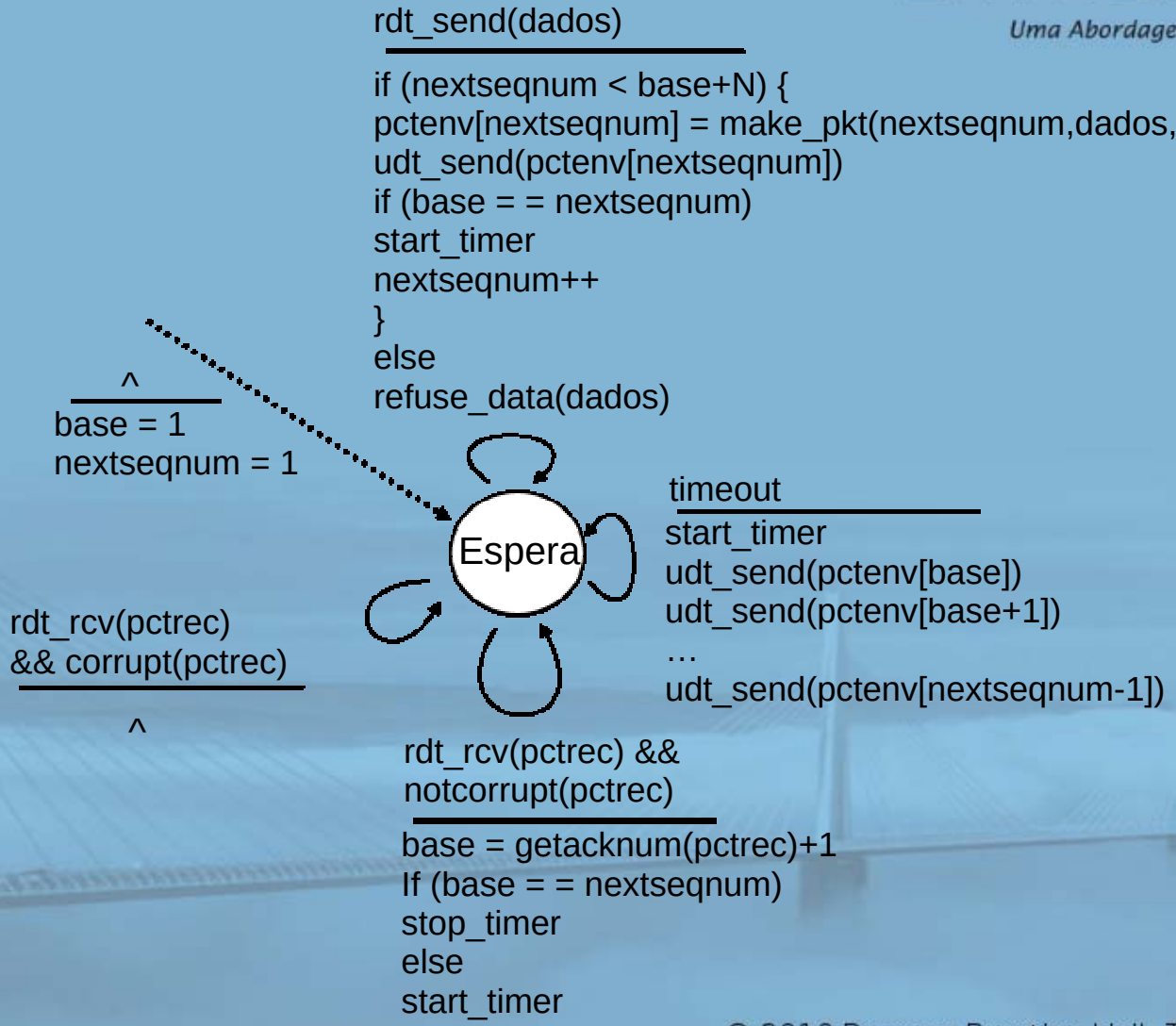
remetente:

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

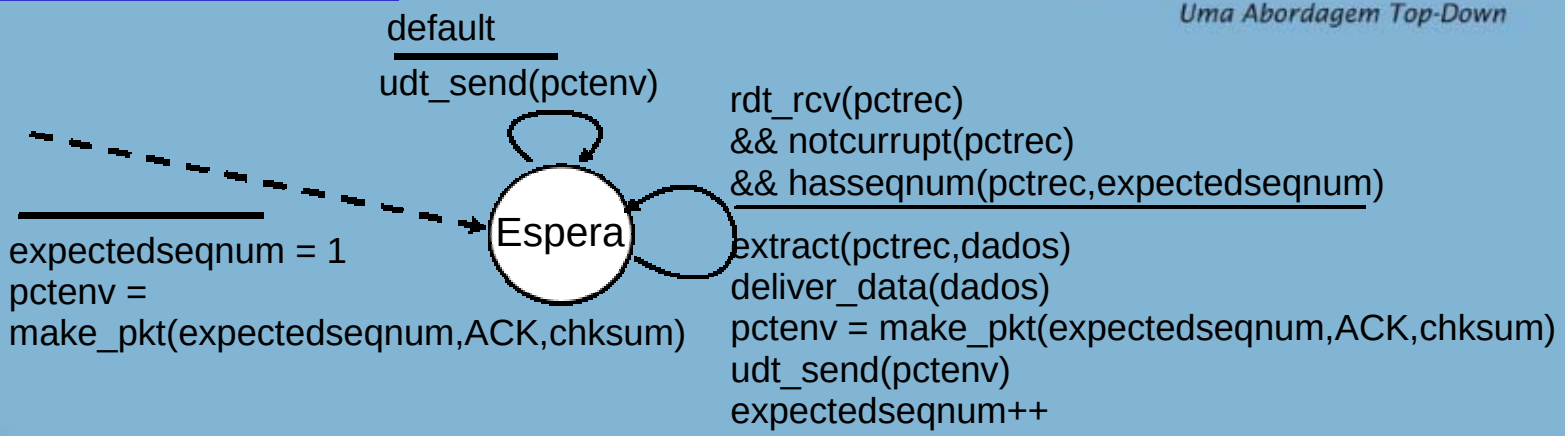


- ACK(n): ACK de todos pcts até inclusive # seq. n – "ACK cumulativo"
  - pode receber ACKs duplicados (ver destinatário)
- temporizador para cada pacote no ar
- *timeout(n):* retransmite pct n e todos pcts com # seq. mais alto na janela

GBN: FSM estendido no remetente



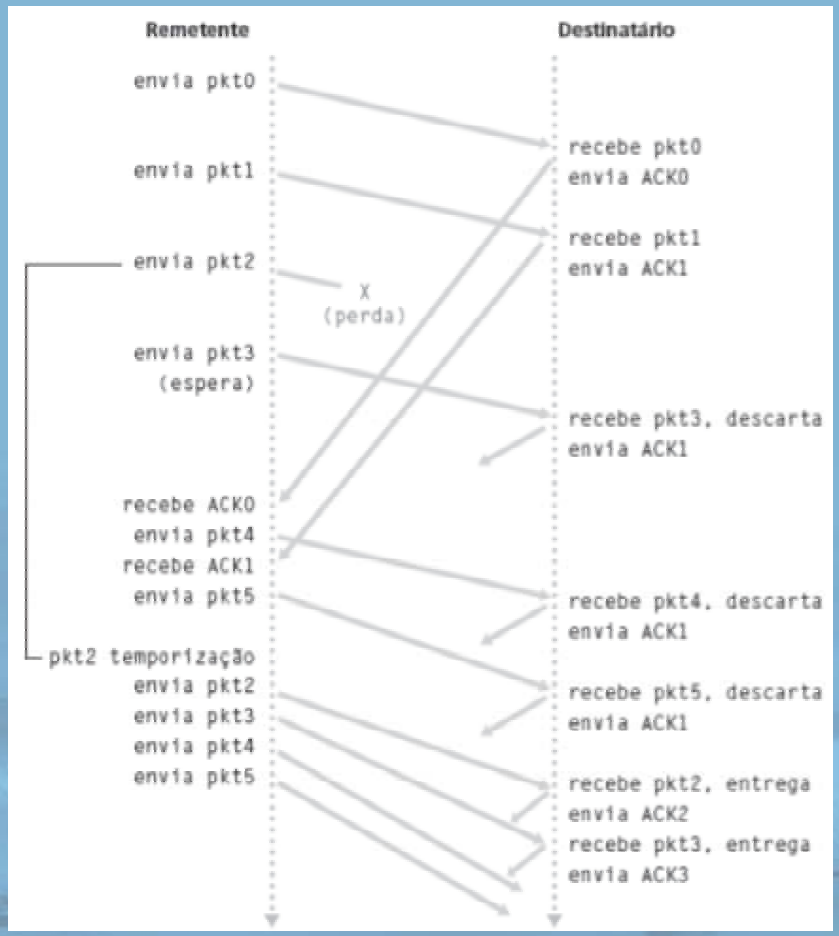
GBN: FSM estendido no destinatário



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

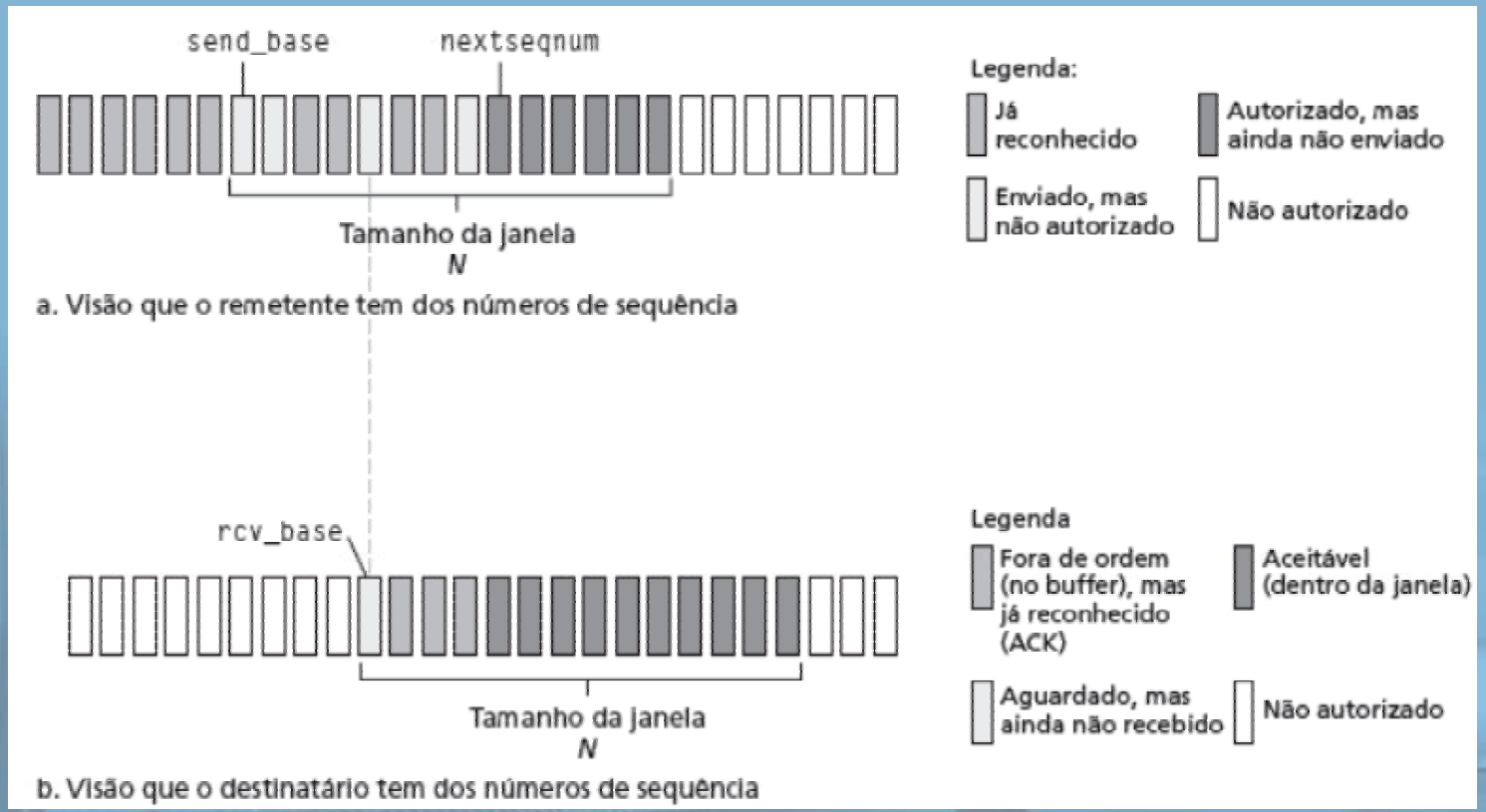
GBN em operação



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

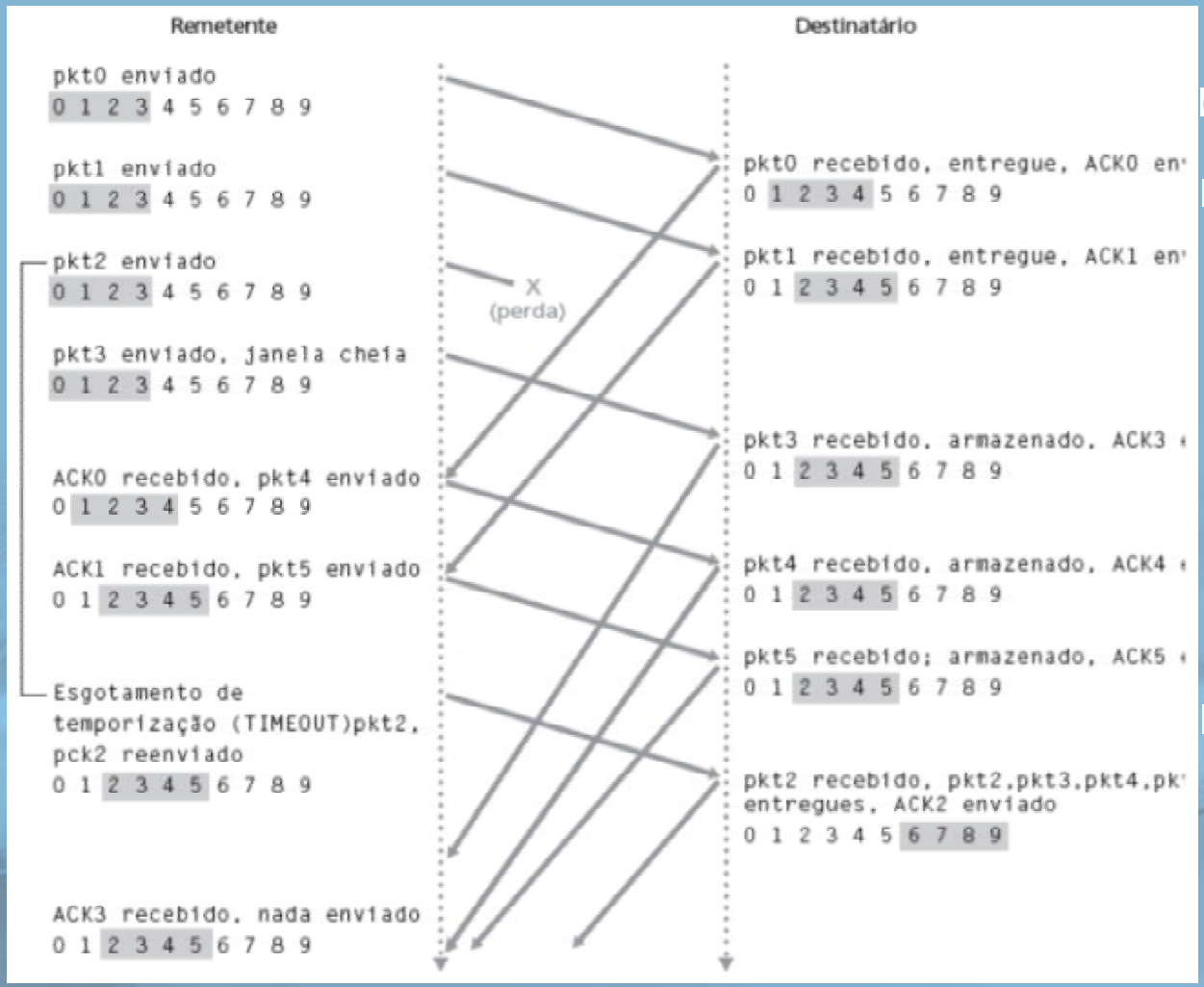
pct n em [rcvbase-N,rcvbase-1]

- ACK(n)

caso contrário:

- ignora

Repetição seletiva em operação



Repetição seletiva: dilema

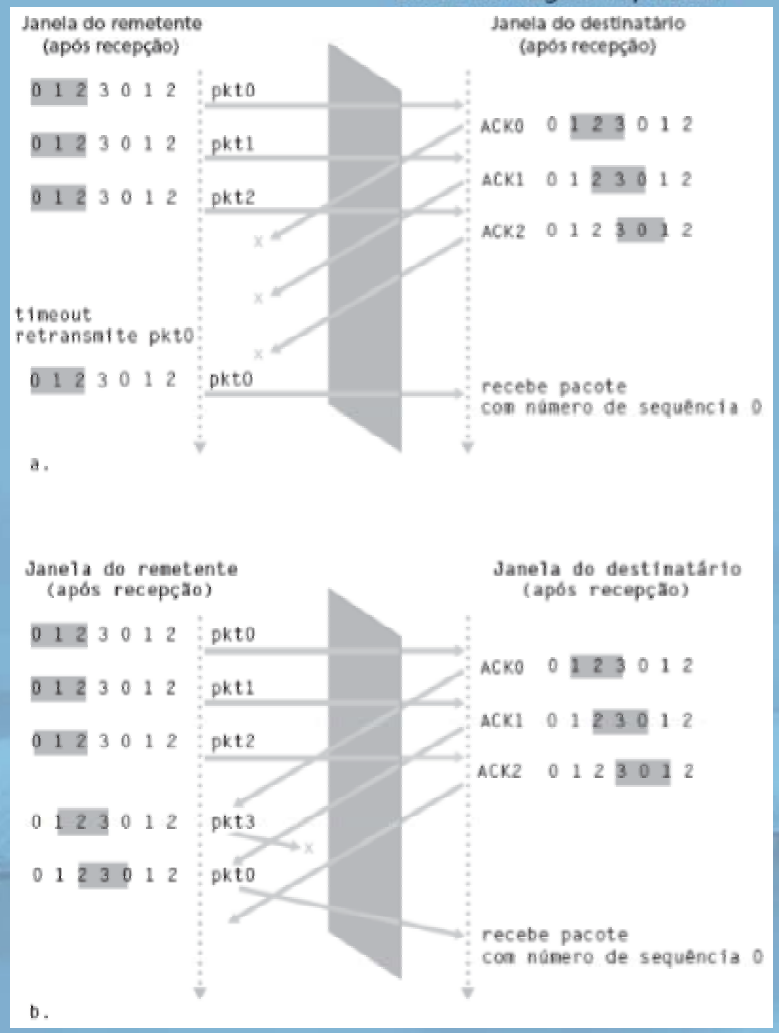
REDES DE COMPUTADORES E A INTERNET 5ª edição

Uma Abordagem Top-Down

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?



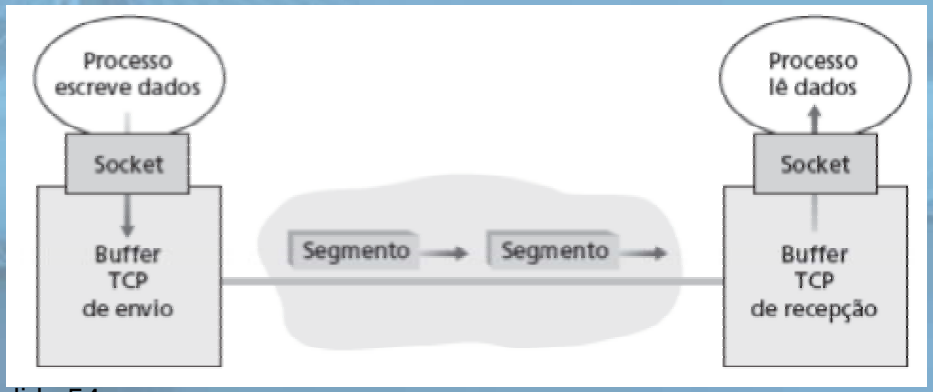
Esboço

- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não orientado para conexão: UDP
- Princípios da transferência confiável de dados
- Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- Princípios de controle de congestionamento
- Congestionamento no TCP

TCP: Visão geral

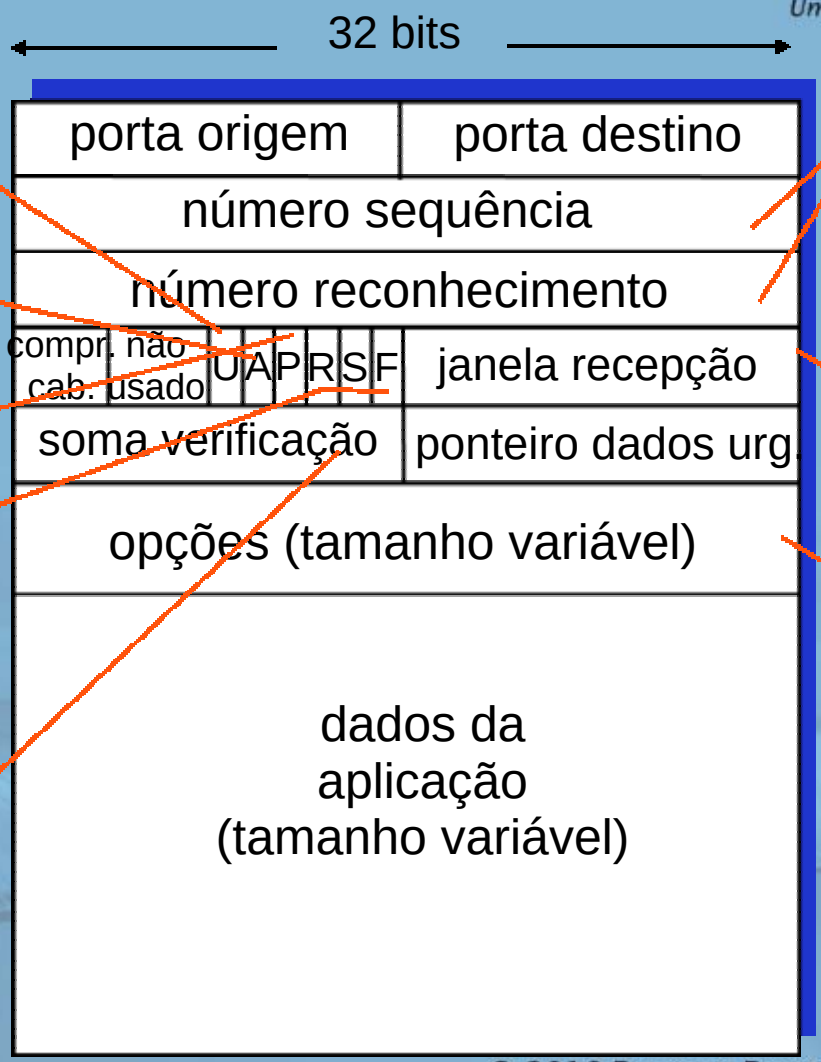
RFCs: 793, 1122, 1323, 2018, 2581

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



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

Estrutura do segmento TCP



URG: dados urgentes (quase não usado)

ACK: # ACK válido

PSH: push dados p/ cima agora (quase não usado)

RST, SYN, FIN: estab. conexão (comandos setup, teardown)

soma de verificação da Internet (como em UDP)

contagem por bytes de dados (não segmentos!)

bytes destinatário pode aceitar

quase sempre vazio,negocia MSS ou opções p/ high speed

#s sequência e ACKs do TCP

#'s de sequência:

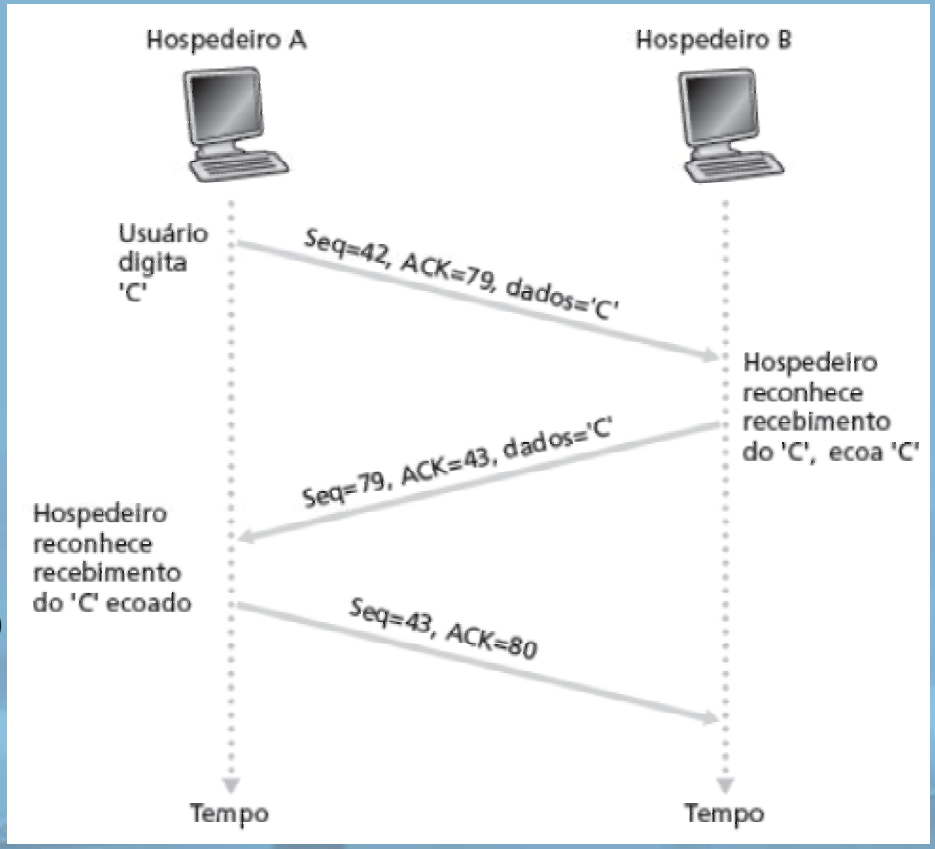
- "número" na cadeia de bytes do 1º byte nos dados do segmento

ACKs:

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

P: como o destinatário trata segmentos fora de ordem

- R: TCP não diz – a critério do implementador



cenário telnet simples

Tempo de ida e volta e *timeout* do TCP

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

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

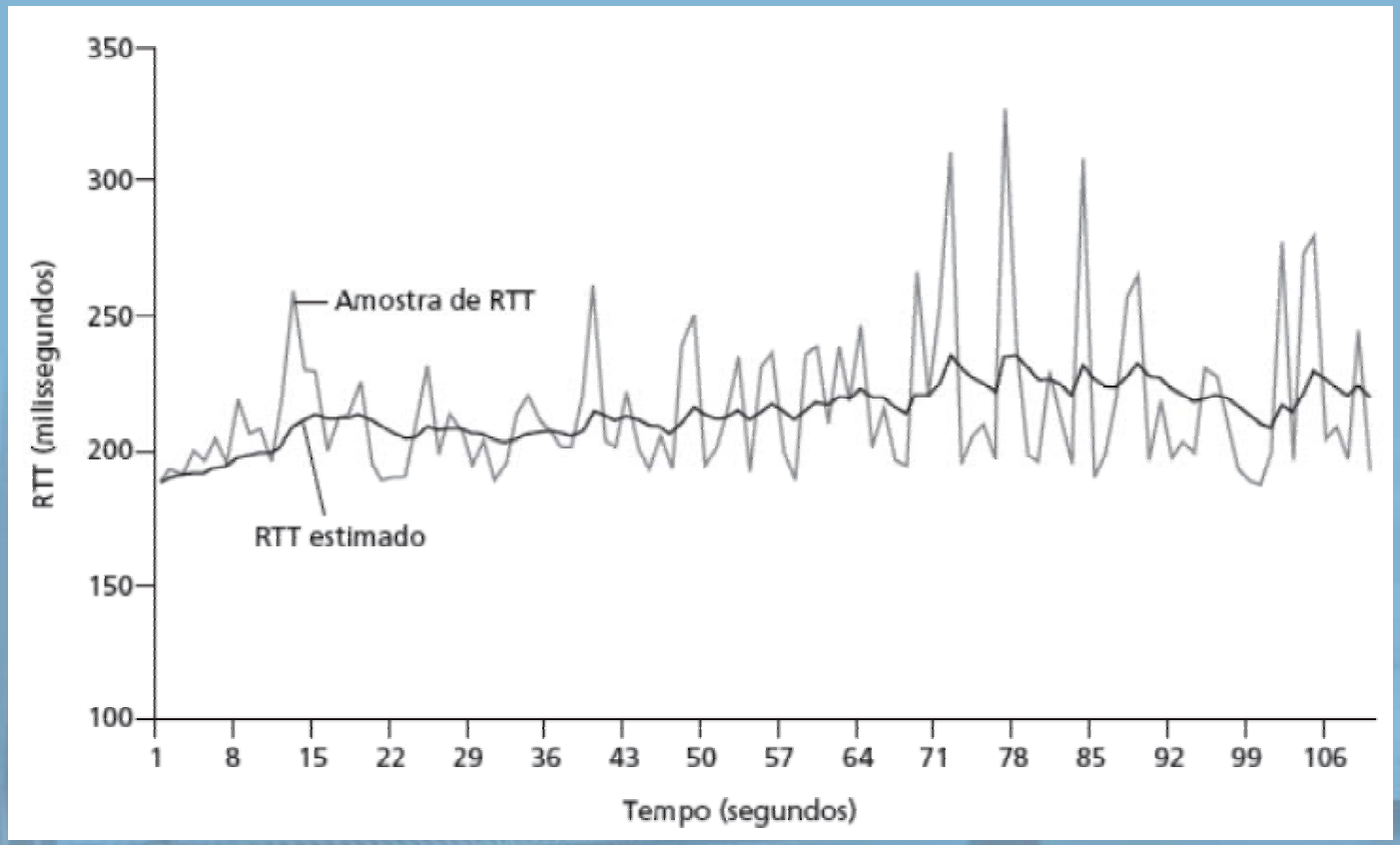
P: Como estimar o RTT?

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

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

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

Amostras de RTTs estimados:



Tempo de ida e volta e *timeout* do TCP

definindo o *timeout*

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

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

(geralmente, α = 0,25)

depois definir intervalo de *timeout*

TimeoutInterval = EstimatedRTT + 4*DevRTT

Transferência confiável de dados no TCP

- TCP cria serviço rdt em cima do serviço não confiável do IP
- segmentos em paralelo
- ACKs cumulativos
- TCP usa único temporizador de retransmissão

- retransmissões são disparadas por:
  - eventos de *timeout*
  - ACKs duplicados
- inicialmente, considera remetente TCP simplificado:
  - ignora ACKs duplicados
  - ignora controle de fluxo, controle de congestionamento

Eventos de remetente TCP:

REDES DE COMPUTADORES E A INTERNET 5ª edição
Uma Abordagem Top-Down

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 */
```

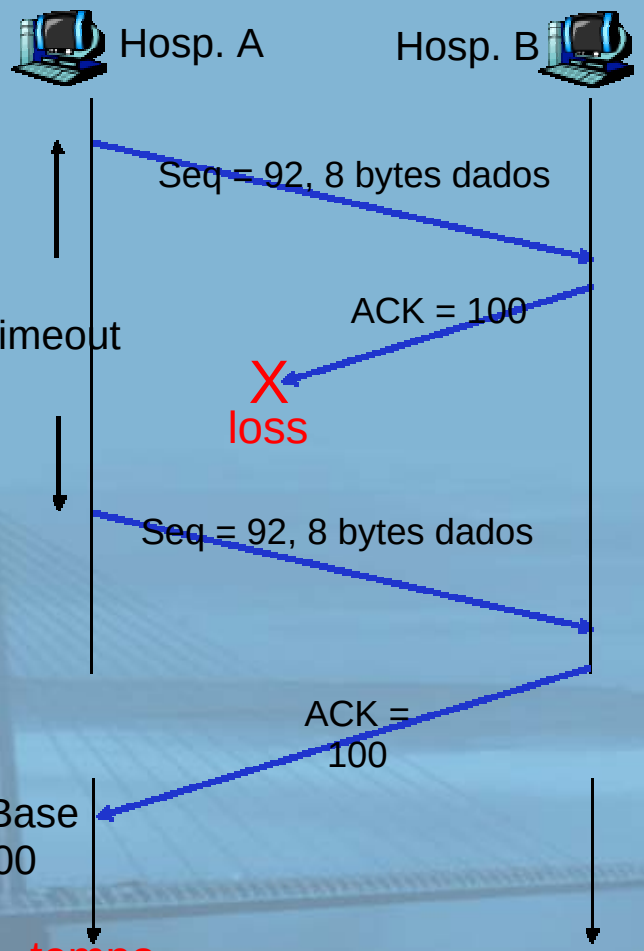
Comentário:

- SendBase-1: último byte cumulativo com ACK

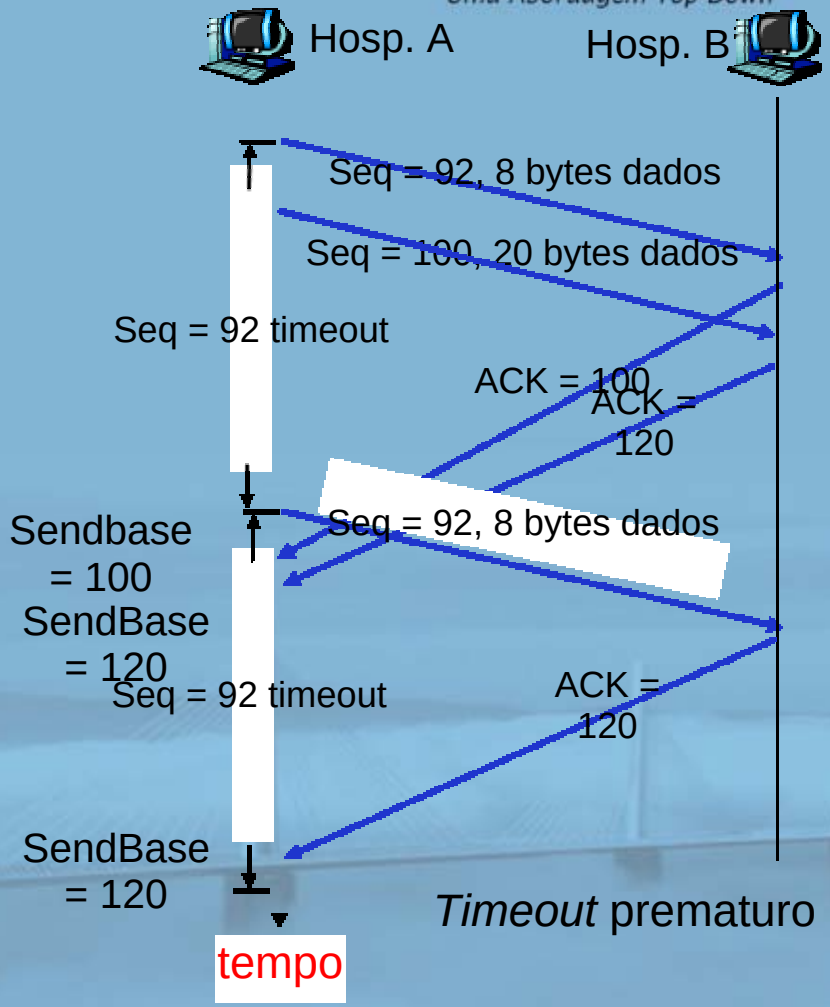
Exemplo:

- SendBase-1 = 71;
  y = 73, de modo que destinatário deseja 73+ ;
  y > SendBase, de modo que novos dados têm ACK

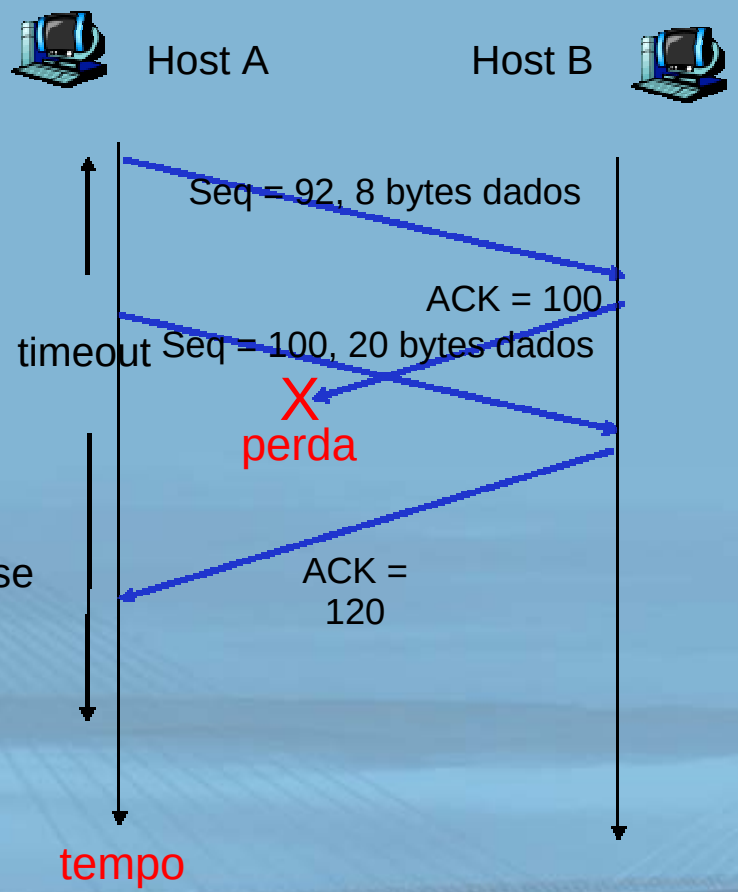
TCP: cenários de retransmissão



Cenário de ACK perdido



Timeout prematuro



Cenário ACK cumulativo

Dobrando *timeout*

- Modificação comum
- Após um *timeout* dobra tempo de time out anterior em vez de usar **EstimtedRTT**

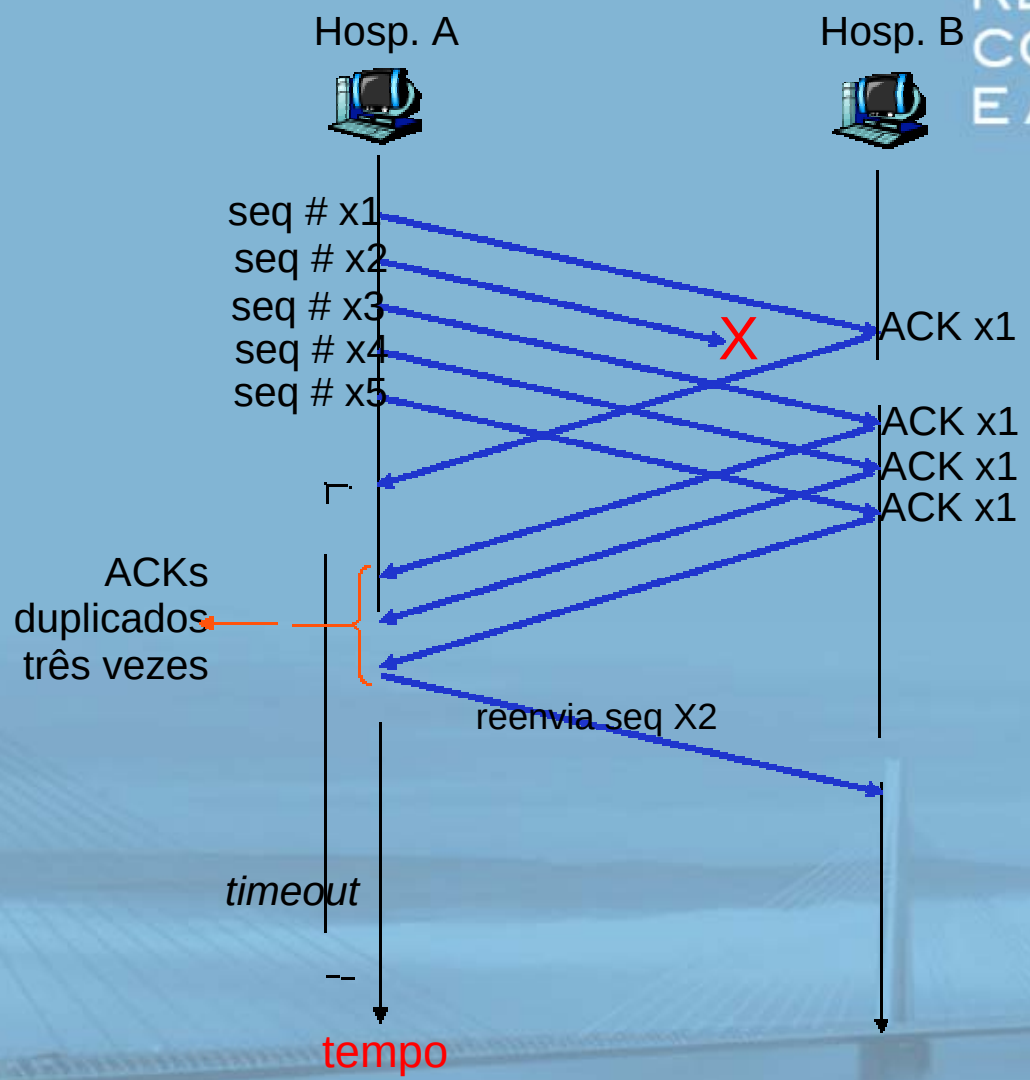
TCP: geração de ACK
[RFC 1122, RFC 2581]

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