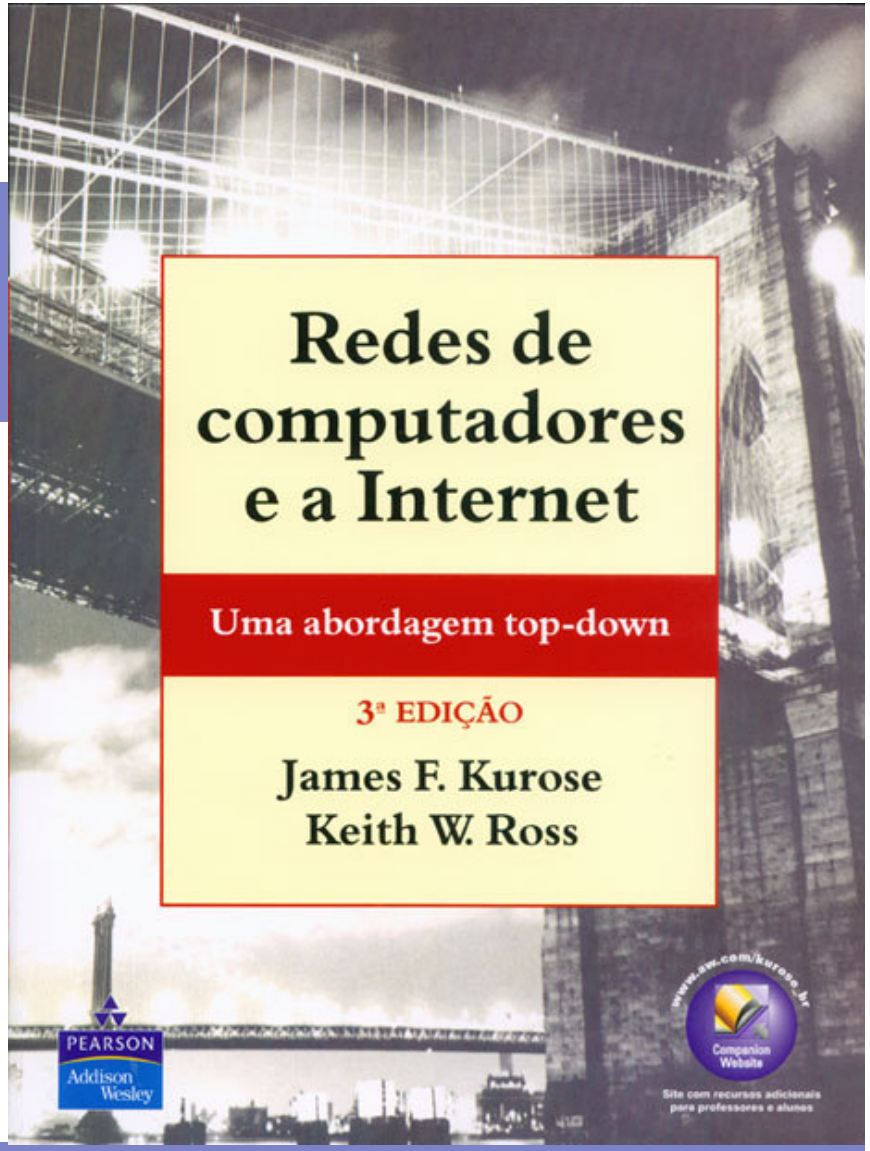
Redes de computadores e a Internet

Prof. Gustavo Wagner

Capítulo 3

Camada de transporte

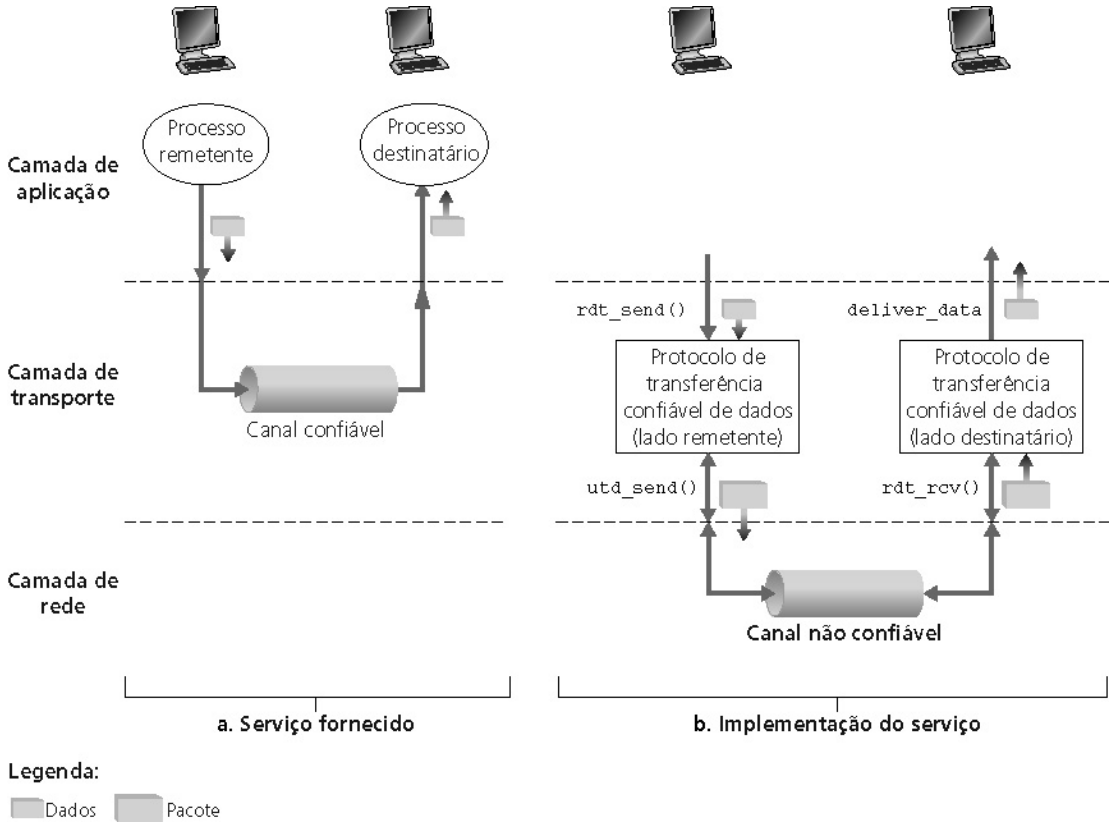


3 Camada de transporte

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

3 Princípios de transferência confiável de dados

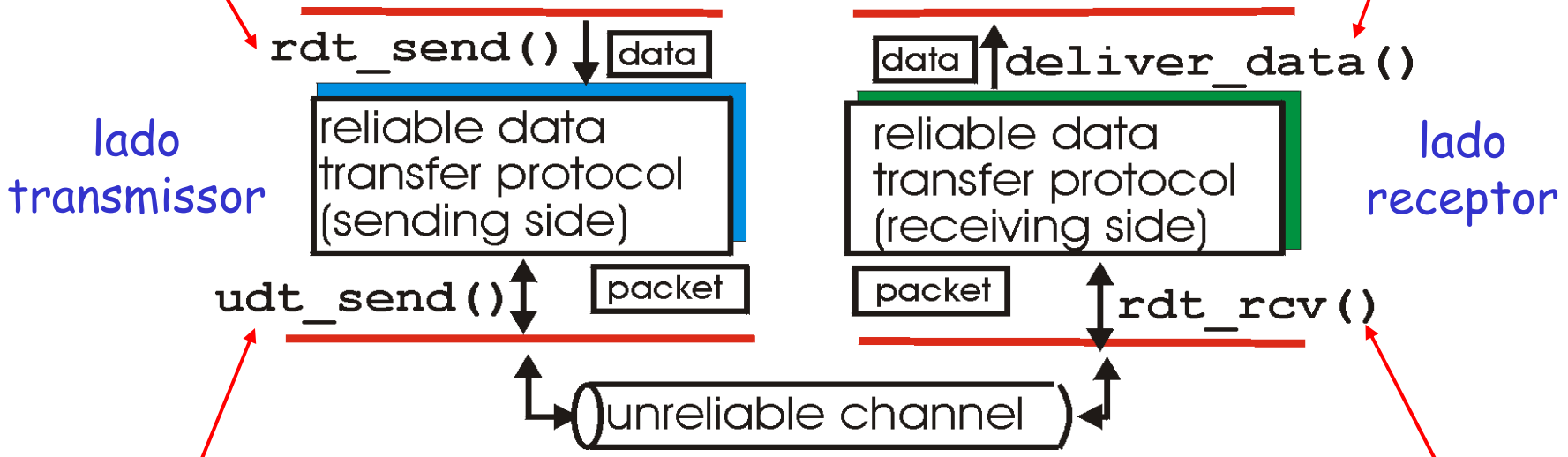
- Importante nas camadas de aplicação, transporte e enlace
- Top 10 na lista dos tópicos mais importants de redes!
- Características dos canais não confiáveis determinarão a complexidade dos protocolos confiáveis de transferência de dados (rdt)



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



udt_send(): chamada pela entidade de transporte, para transferir pacotes para o receptor sobre o canal não confiável

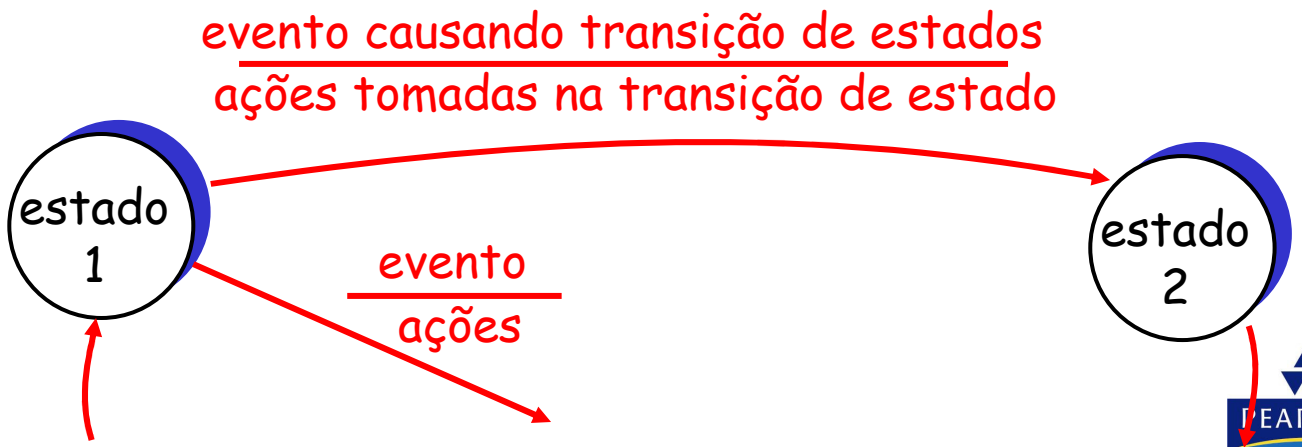
rdt_rcv(): chamada quando o pacote chega ao lado receptor do canal

3 Transferência confiável: o ponto de partida

Etapas:

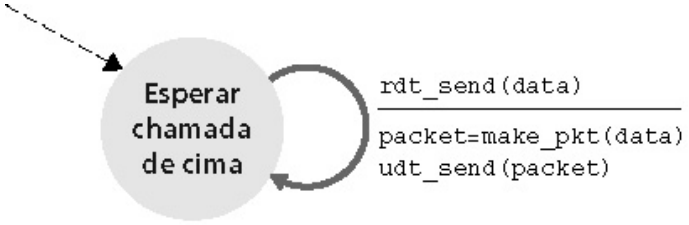
- Desenvolver incrementalmente o transmissor e o receptor de um protocolo confiável de transferência de dados (rdt)
- Considerar apenas transferências de dados unidirecionais
  - Mas informação de controle deve fluir em ambas as direções!
- Usar máquinas de estados finitos (FSM) para especificar o protocolo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

estado: quando neste "estado" o próximo estado fica unicamente determinado pelo próximo evento

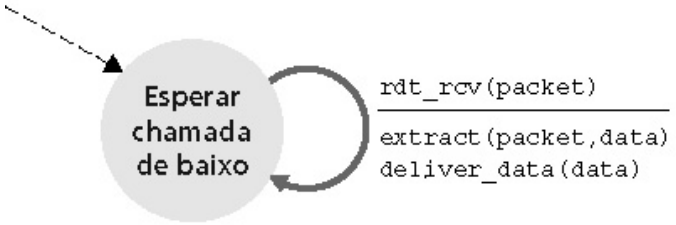


3 rdt1.0: Transfêrencia confiável sobre canais confiáveis

- 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

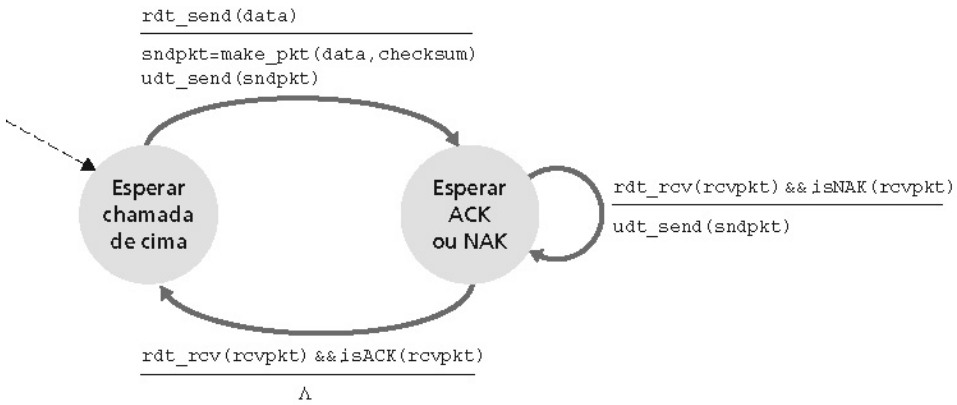


b. rdt1.0: lado destinatário

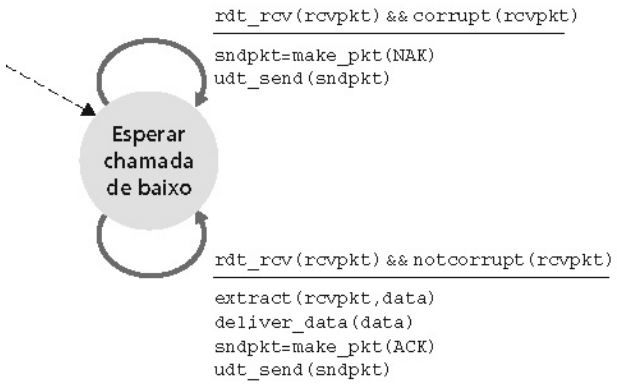
3 rdt2.0: canal com erros de bit

- 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 FSM

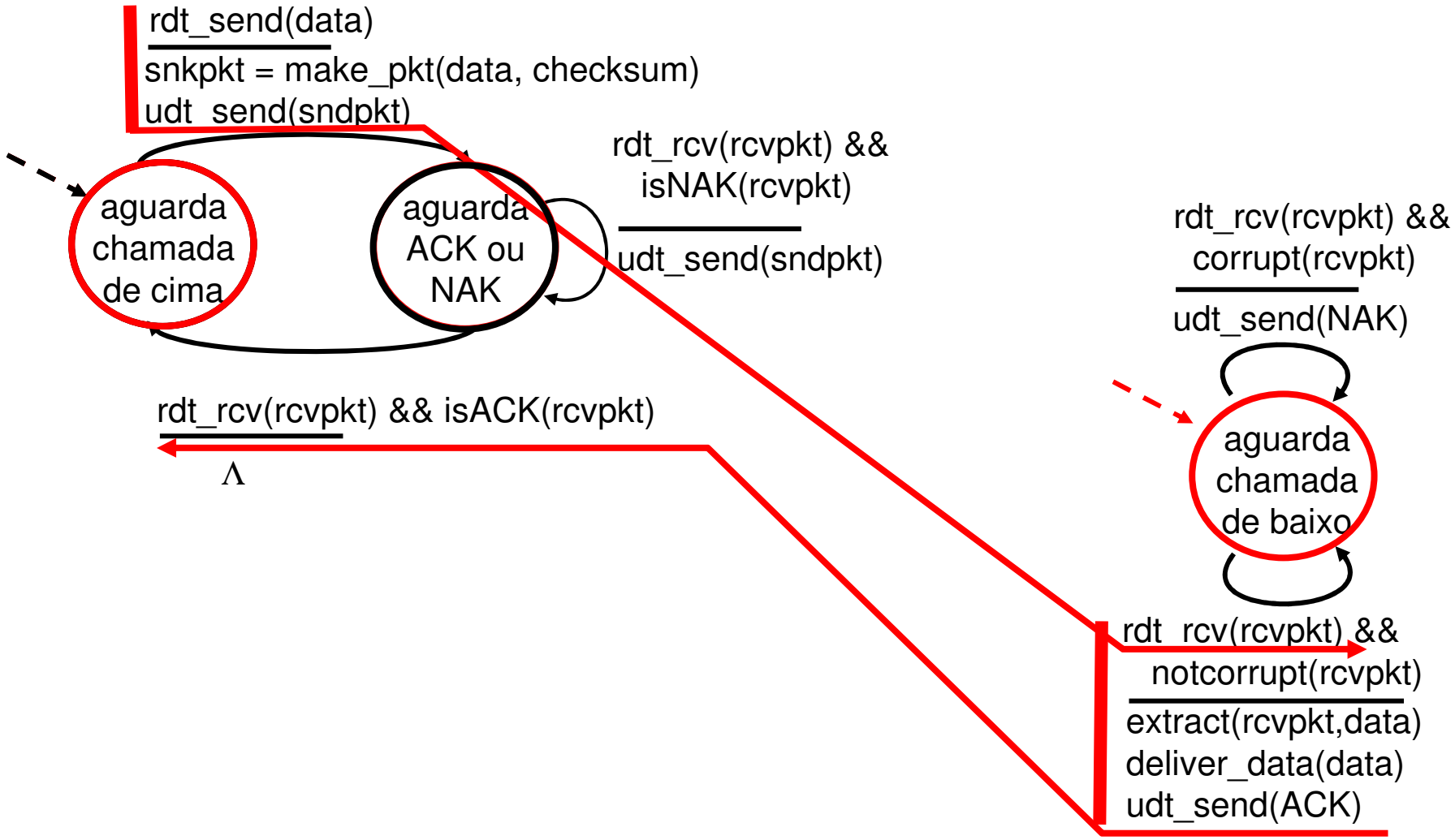


a. rdt2.0: lado remetente

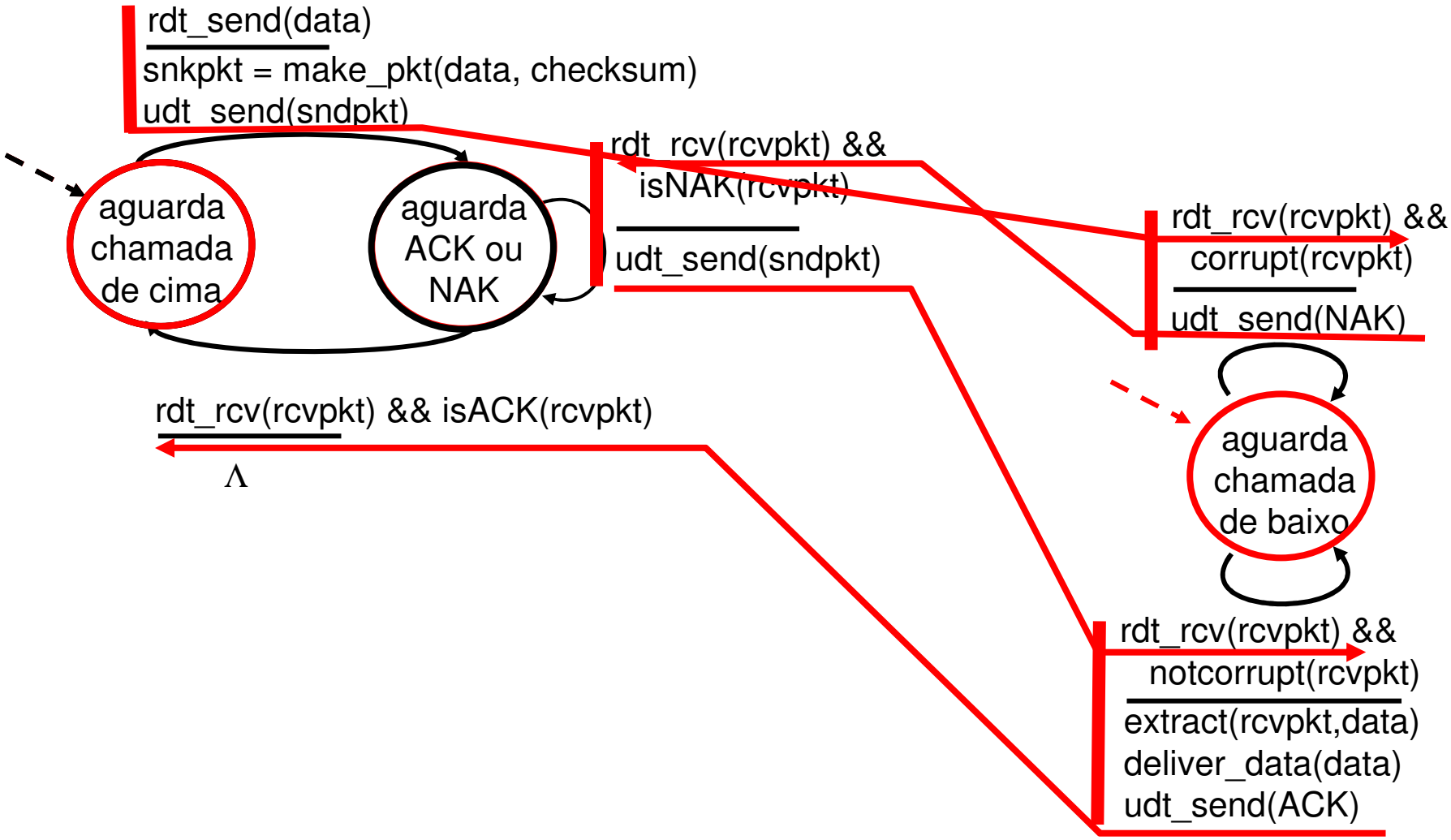


b. rdt2.0: lado destinatário

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

Possíveis soluções:

- Transmissor pergunta o que receptor disse;
  - Problema: e se o receptor achar que a pergunta do transmissor é uma nova mensagem, e não pedido de reenvio?
- Adicionar número suficiente de bits de soma de verificação, fazendo que remetente possa detectar e recuperar o erro
  - Isso resolve para canais que podem corromper pacotes, não para canais que podem perder pacotes!);
- Transmissor reenviar o pacote de dados quando receber ACK ou NAK truncado;
  - Problema: essa solução acrescenta pacotes duplicados

3 rdt2.0 tem um problema fatal!

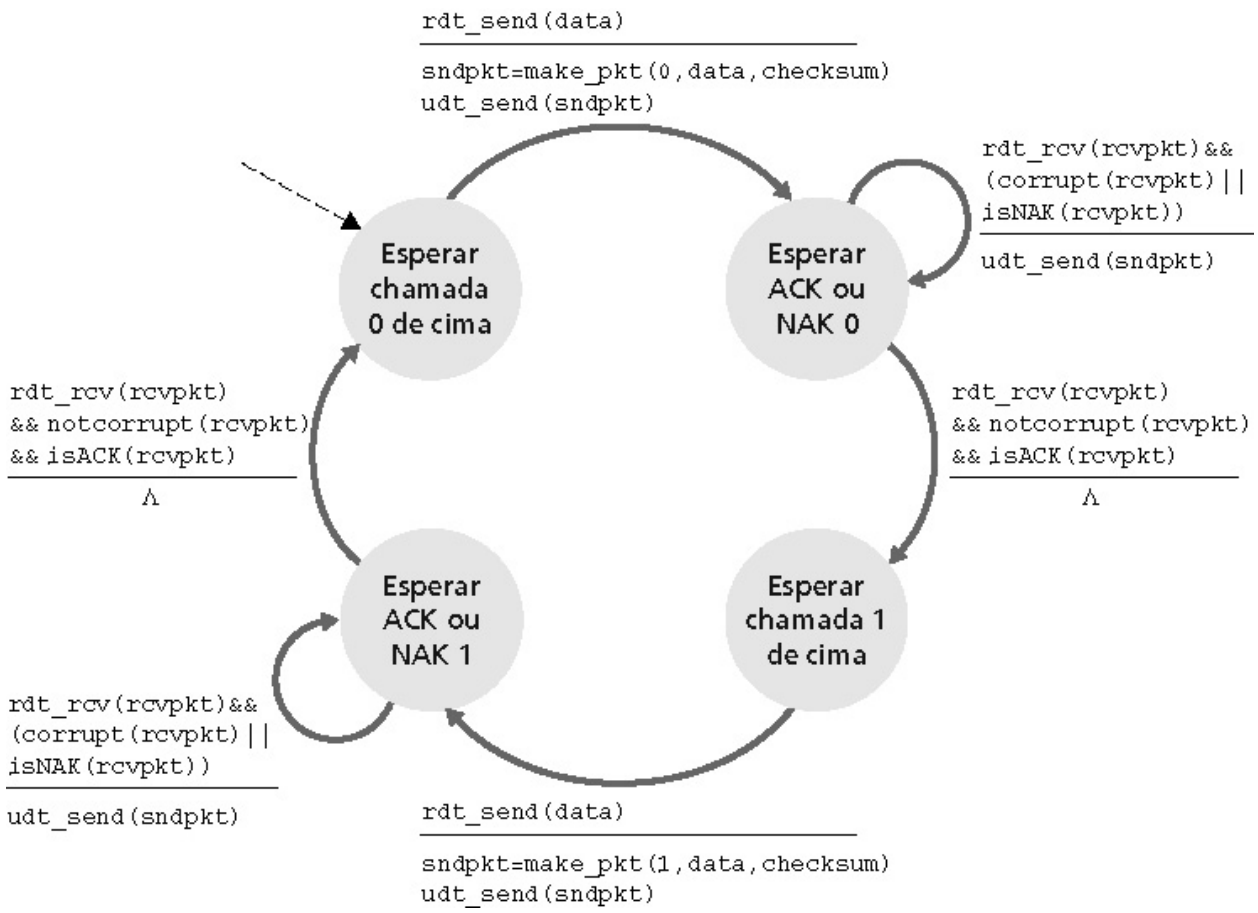
Tratando duplicatas:

- 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

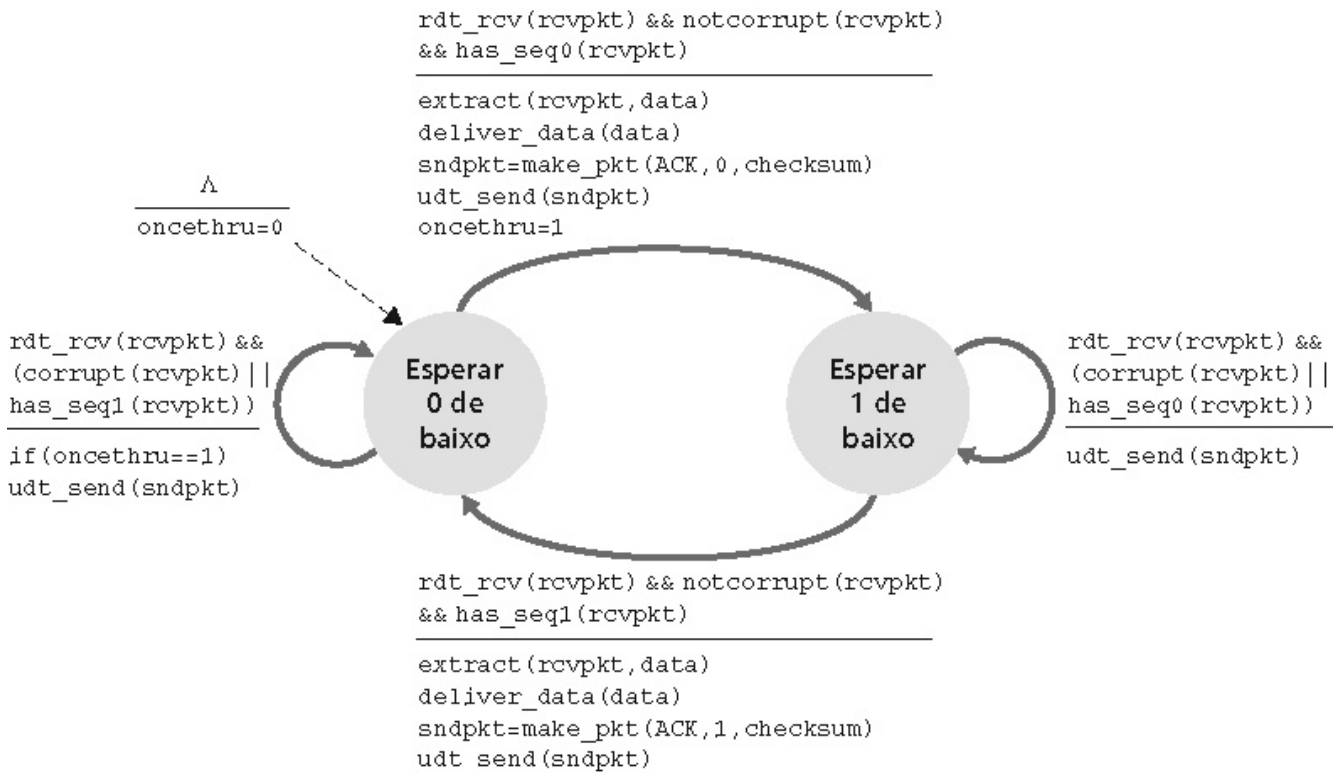
Tipo de protocolo Pare e espere

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

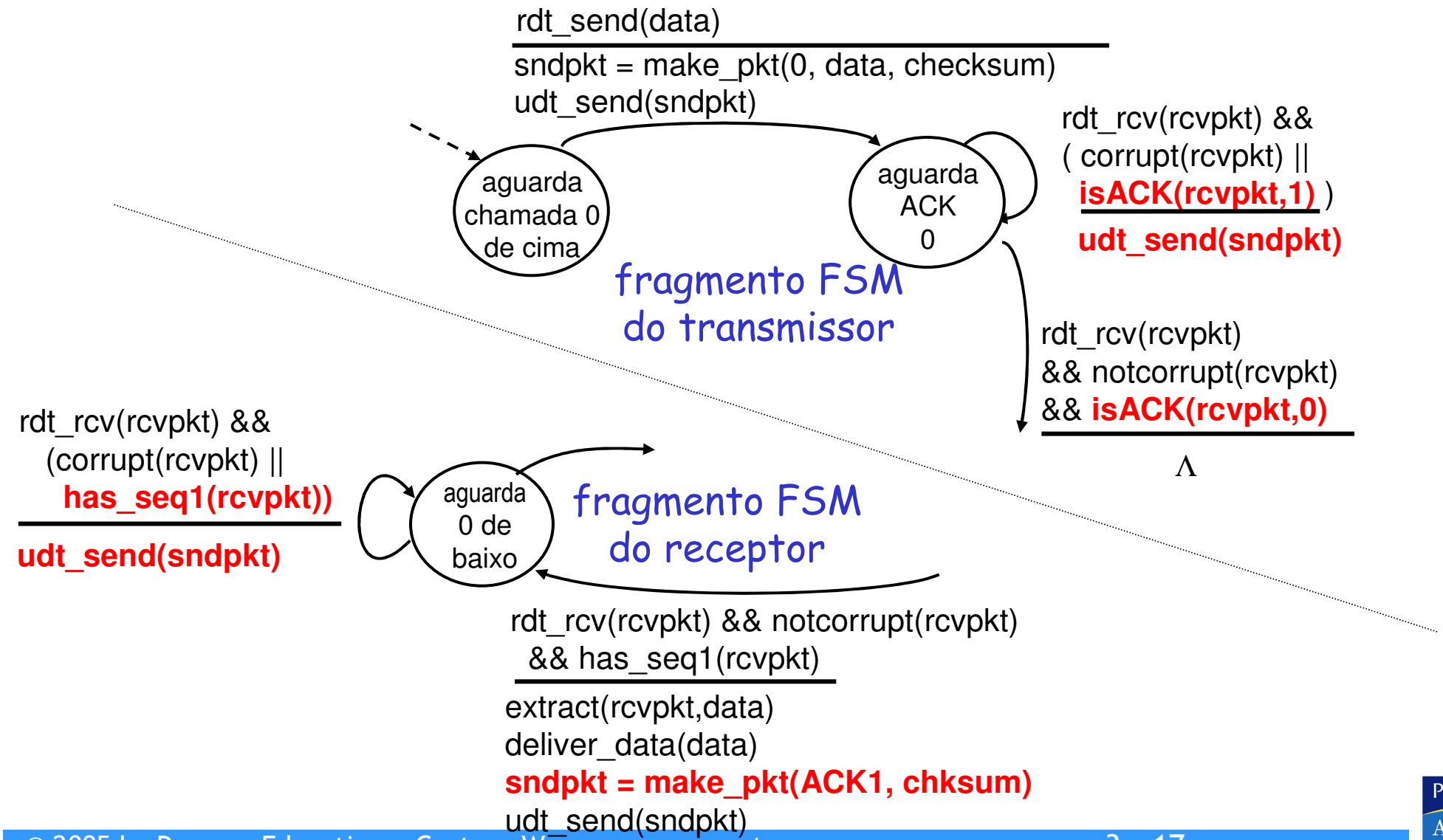
Receptor:

- 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

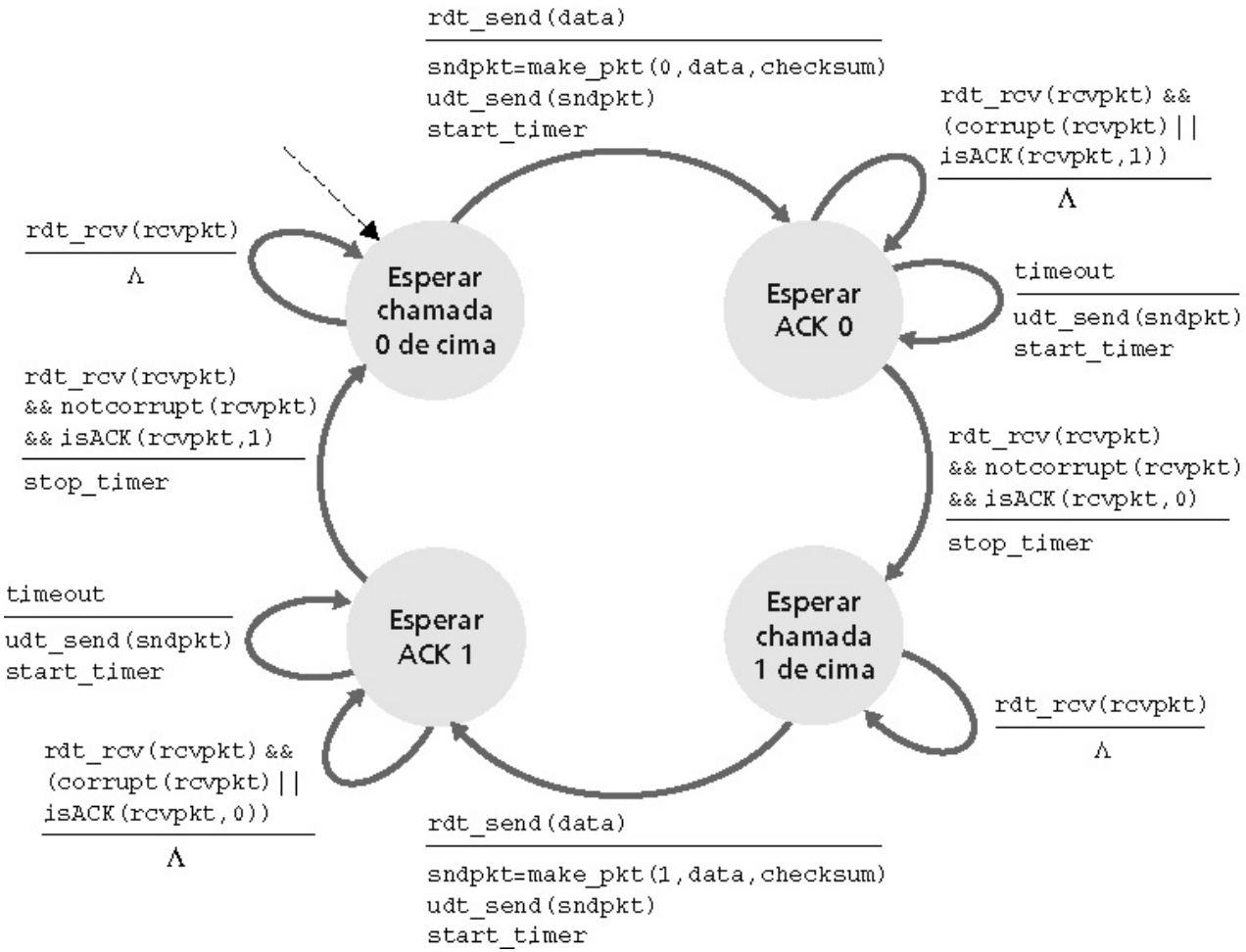
Nova hipótese: canal de transmissão pode também perder pacotes (dados ous ACKs)

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

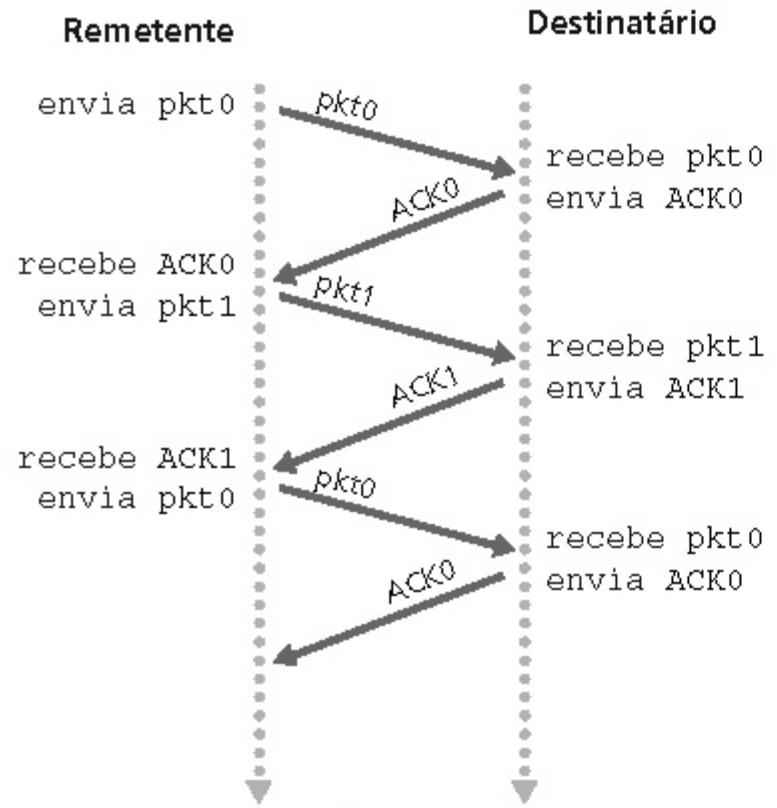
Abordagem: transmissor espera um tempo "razoável" pelo ACK

- 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

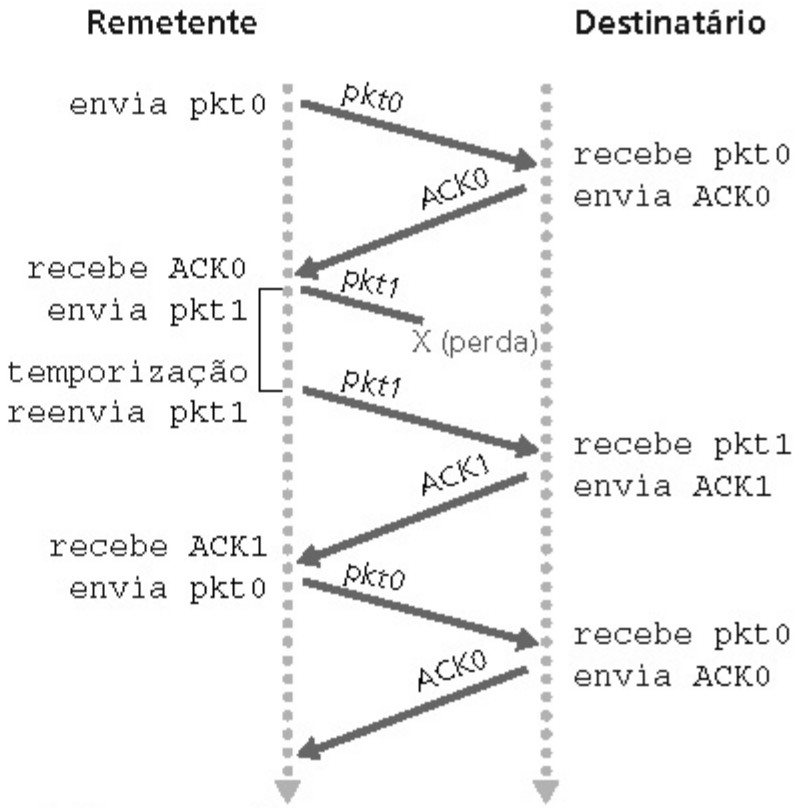
3 Transmissor rdt3.0



3 rdt3.0 em ação

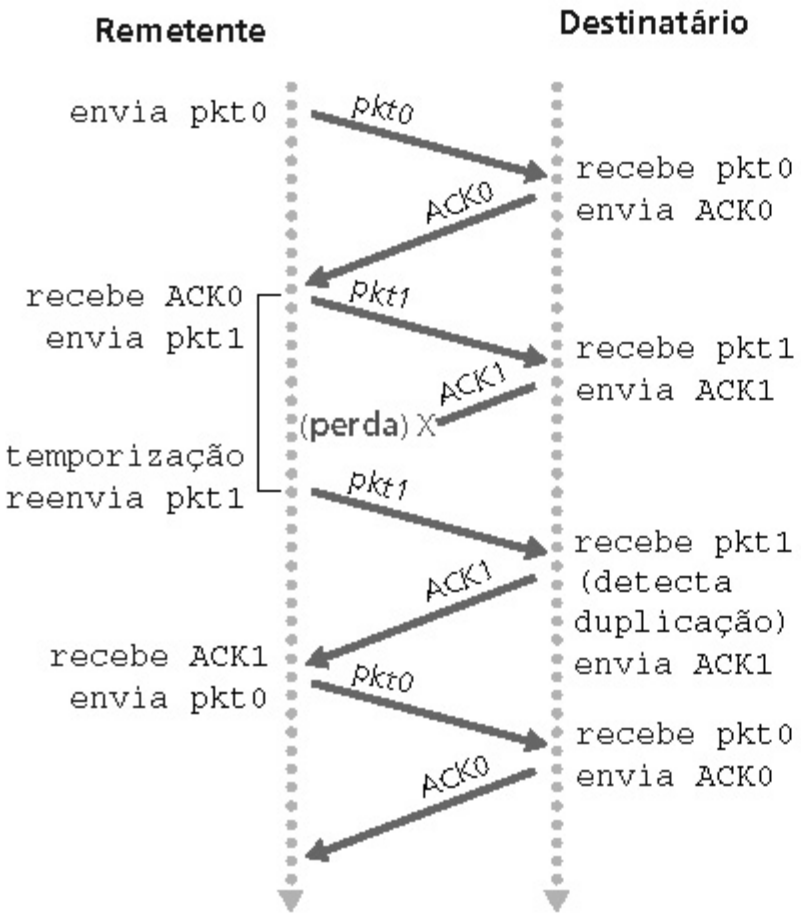


a. Operação sem perda

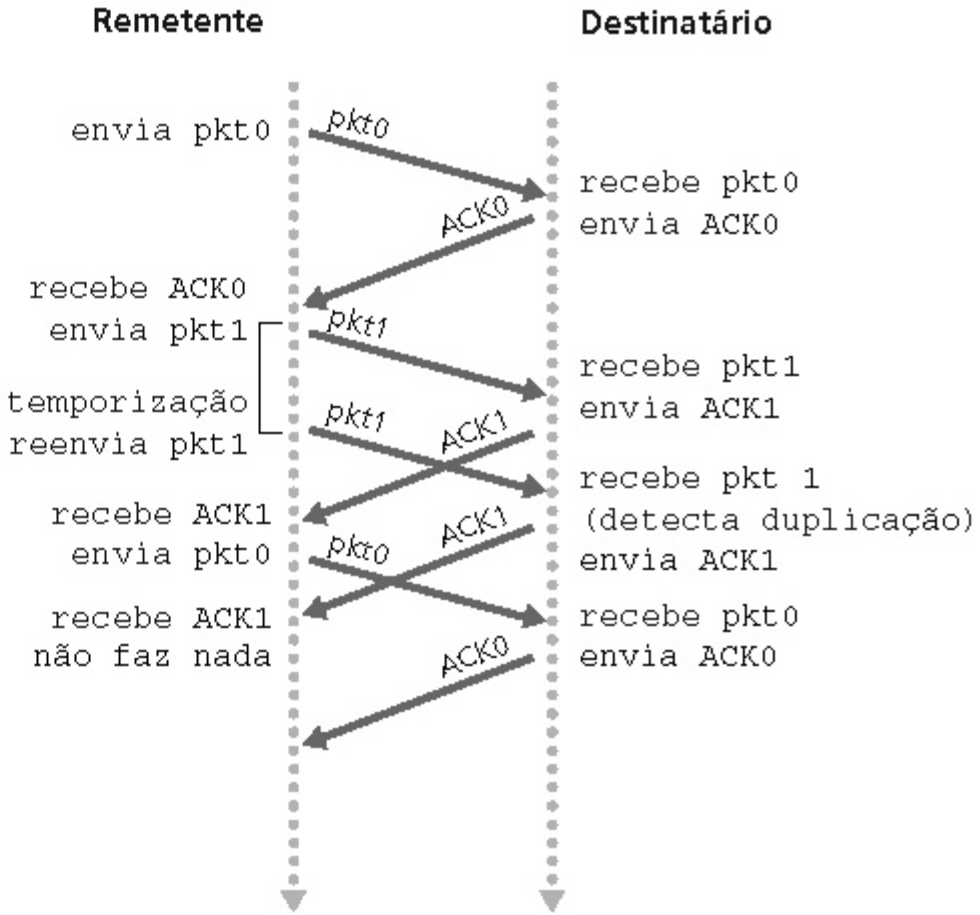


b. Pacote perdido

3 rdt3.0 em ação



c. ACK perdido



d. Interrupção prematura

3 Desempenho do rdt3.0

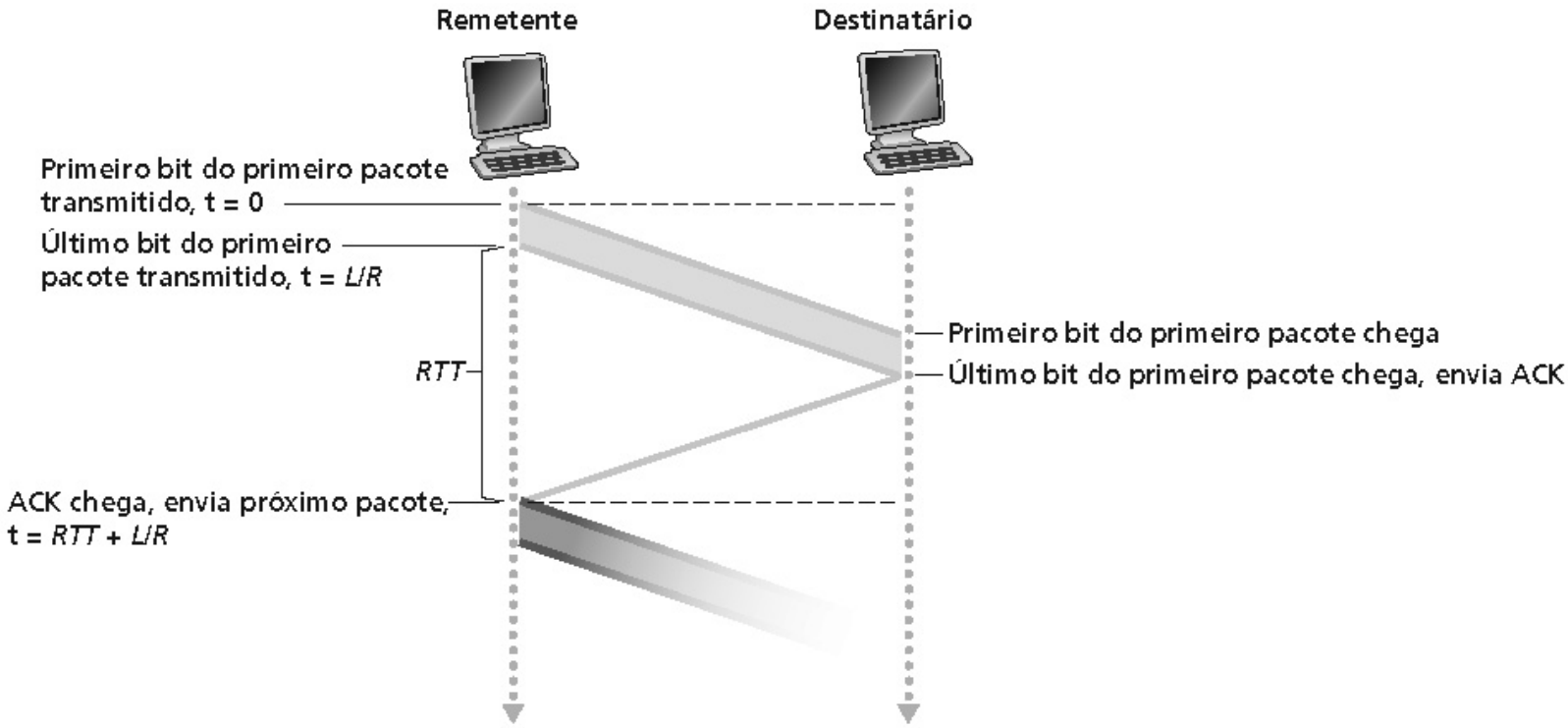
- rdt3.0 funciona, mas o desempenho é sofrível
- Exemplo: enlace de 1 Gbps, 15 ms de atraso de propagação, pacotes de 1 KB:

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

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027\ \text{microseg}$$

- U_{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*



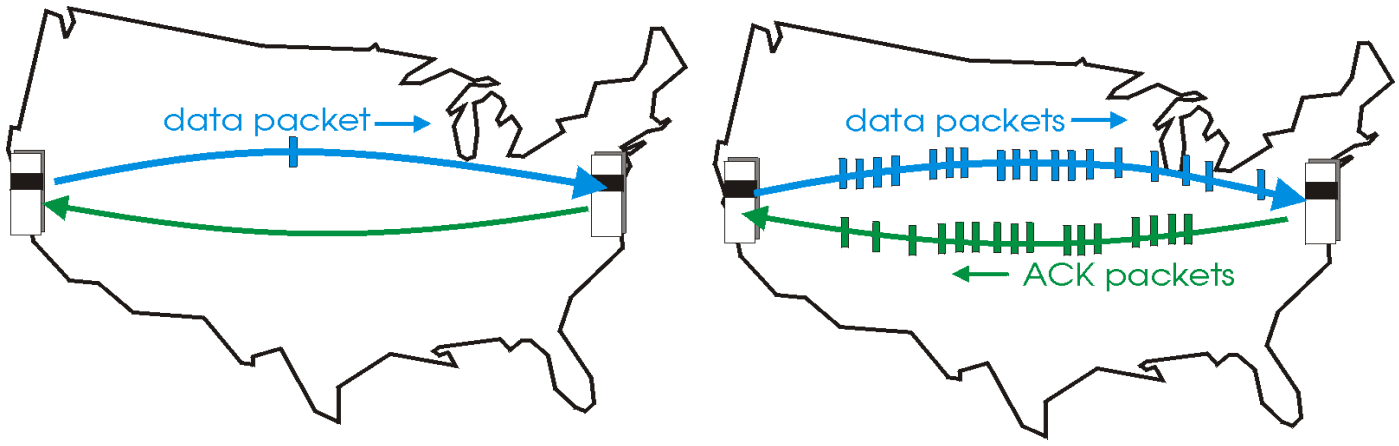
a. Operação pare e espere

$$\text{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

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

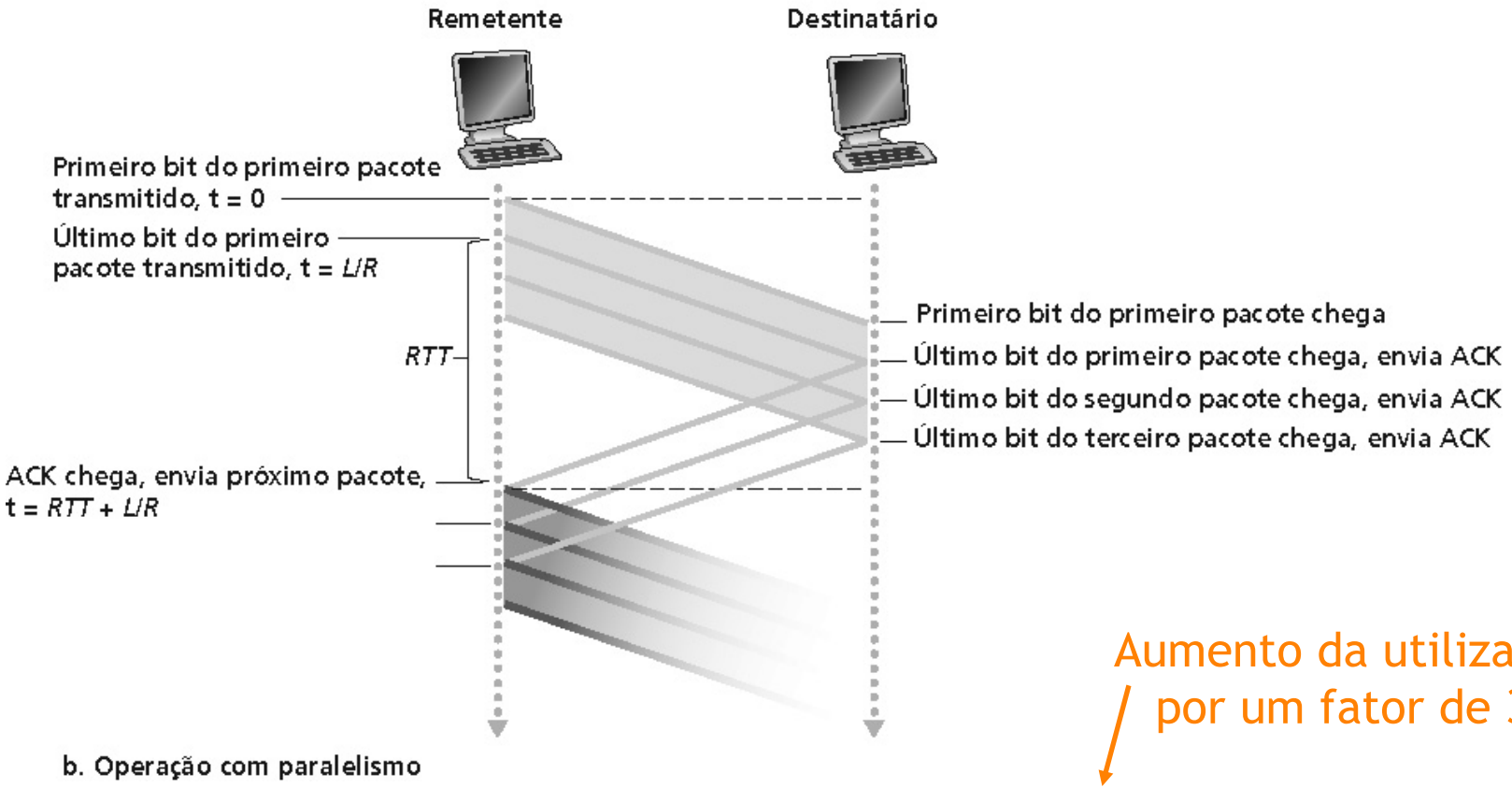


(a) operação do protocolo pare e espere

(a) operação do protocolo com paralelismo

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

3 Pipelining: aumento da utilização



b. Operação com paralelismo

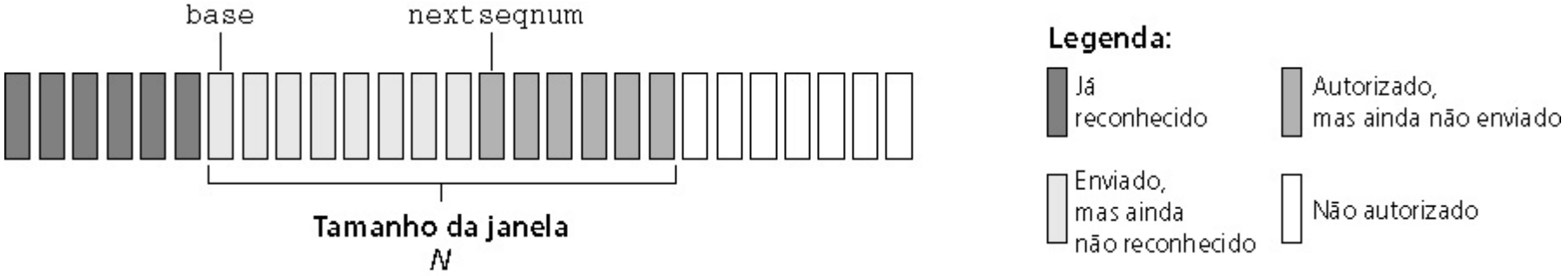
Aumento da utilização por um fator de 3!

$$U_{sender} = \frac{3 * L/R}{RTT + L/R} = \frac{.024}{30.008} = 0.0008$$

3 Go-Back-N

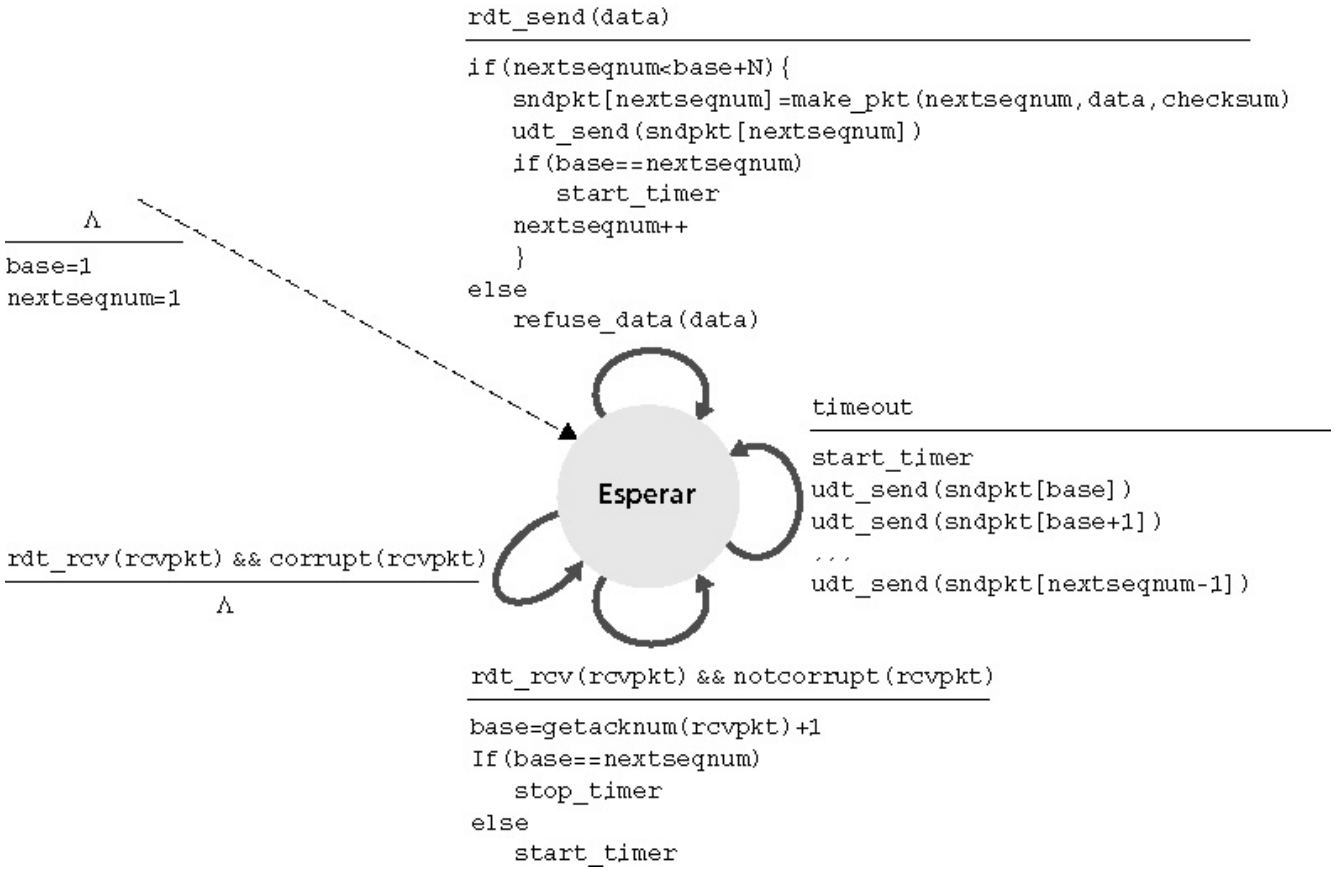
Transmissor:

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

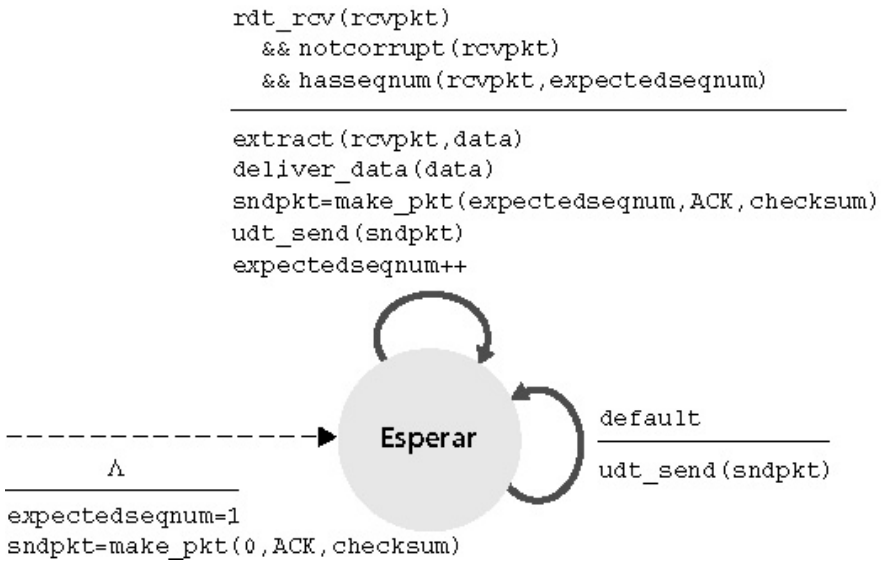


- ACK(n): reconhece todos os pacotes até o número de seqüência N (incluindo este limite). “ACK cumulativo”
  - Pode receber ACKs duplicados (veja receptor)
- Temporizador para cada pacote enviado e não confirmado
- **Tempo de confirmação (n)**: retransmite pacote n e todos os pacotes com número de seqüência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

3 GBN: FSM estendida para o transmissor

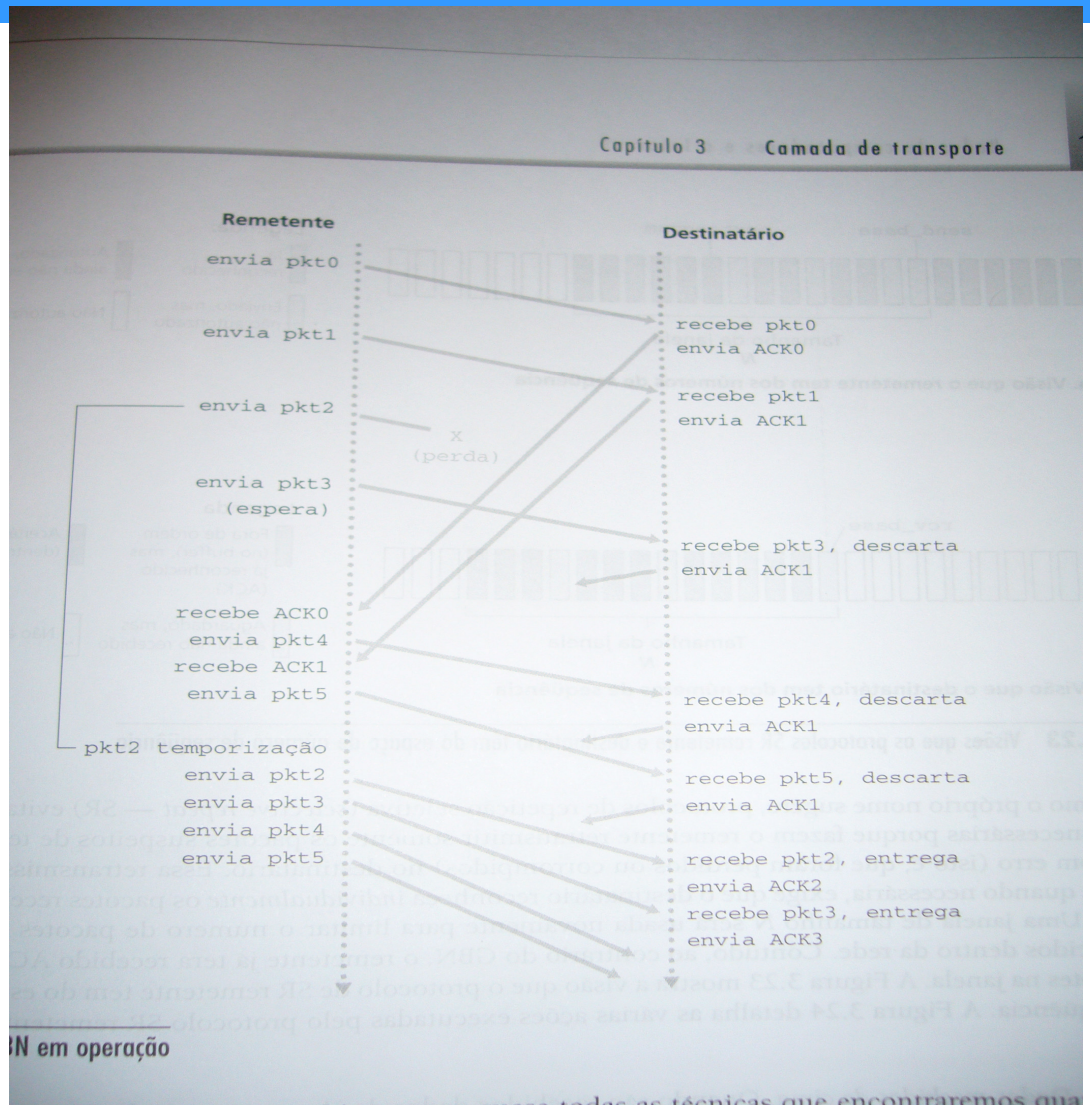


3 GBN: FSM estendida para o receptor



- 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

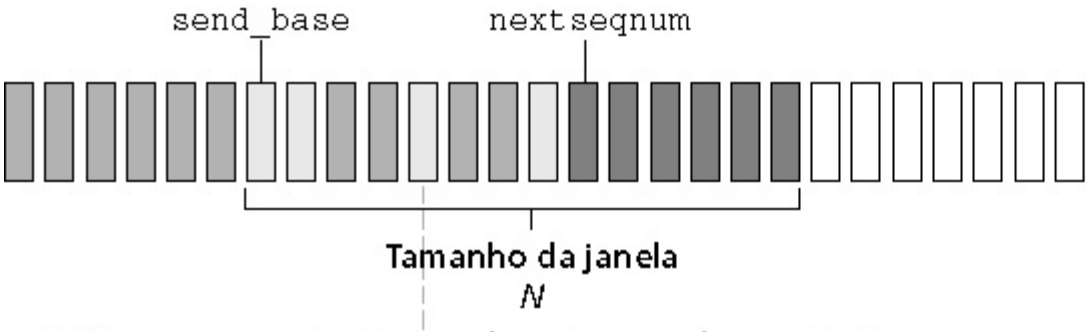


Applet

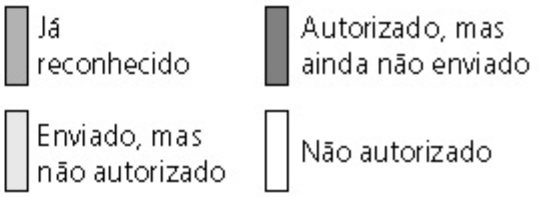
3 Retransmissão seletiva

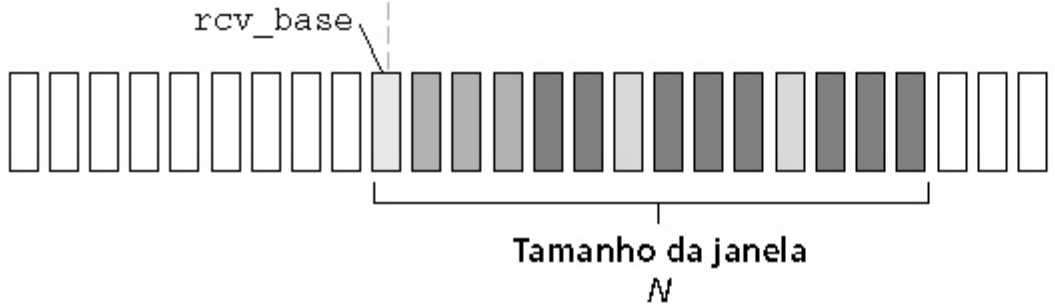
- Receptor reconhece **individualmente** todos os pacotes recebidos corretamente
  - Armazena pacotes, quando necessário, para eventual entrega em ordem para a camada superior
- Transmissor somente reenvia os pacotes para os quais um ACK não foi recebido
  - Transmissor temporiza cada pacote não reconhecido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

3 Retransmissão seletiva: janelas do transmissor e do receptor

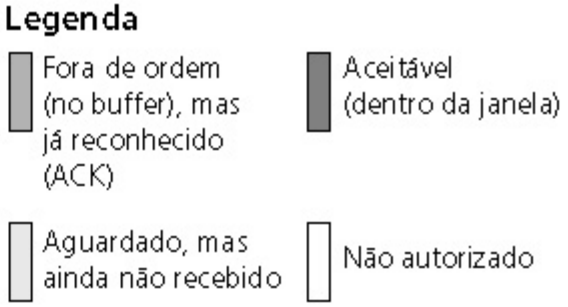


a. Visão que o remetente tem dos números de seqüência





b. Visão que o destinatário tem dos números de seqüência



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

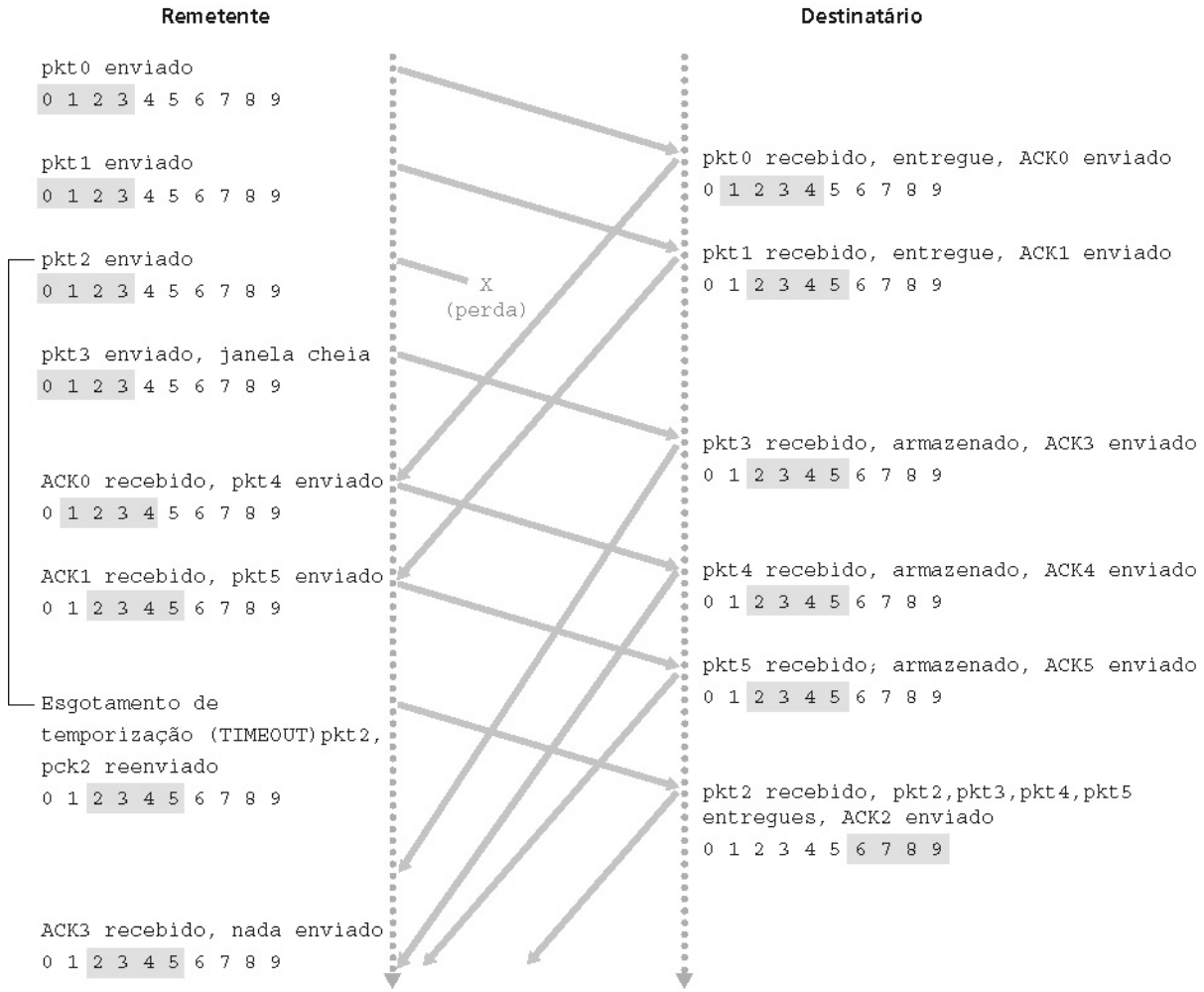
pkt n em [rcvbase-N,rcvbase-1]

- ACK(n)

Caso contrário:

- Ignora

3 Retransmissão seletiva em ação



3 Retransmissão seletiva: dilema

Exemplo:

- Seqüências: 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?