Redes de computadores II

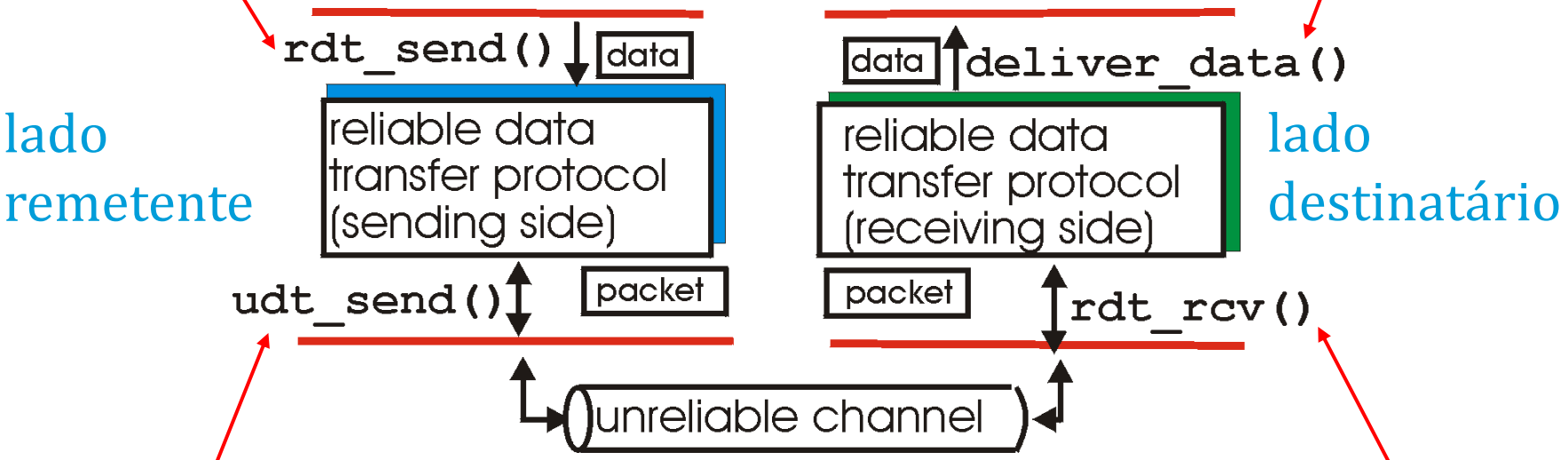
Aula 6 – Princípios da Transferência confiável.

Prof. Felipe Leite Lobo
felipe.lobo@ufrr.br

Transferência confiável de dados

`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



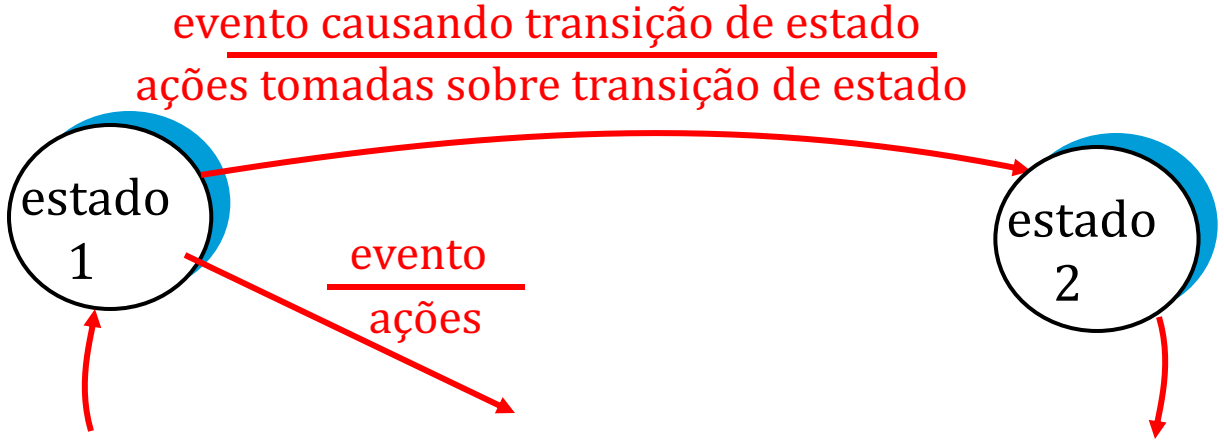
`udt_send()`: chamado pela rdt, para transferir pacote por canal não confiável ao destinatário

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

Transferência confiável de dados

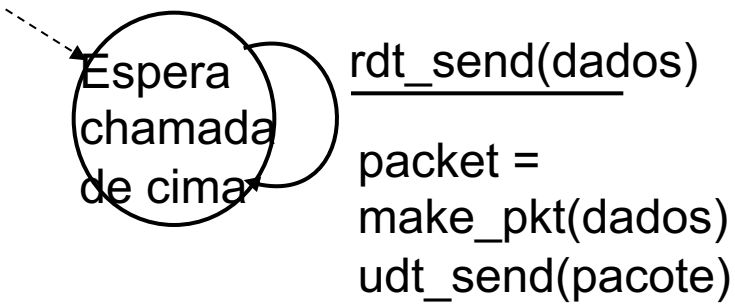
- Desenvolver de forma incremental os lados remetente e destinatário do protocolo de transferência confiável de dados (rdt);
- Considerar apenas a transferência de dados unidirecional;
  - mas informações de controle fluirão nas duas direções!
- Usar máquinas de estado finito (FSM) para especificar remetente, destinatário;

estado: quando neste "estado", próximo estado determinado exclusivamente pelo próximo evento

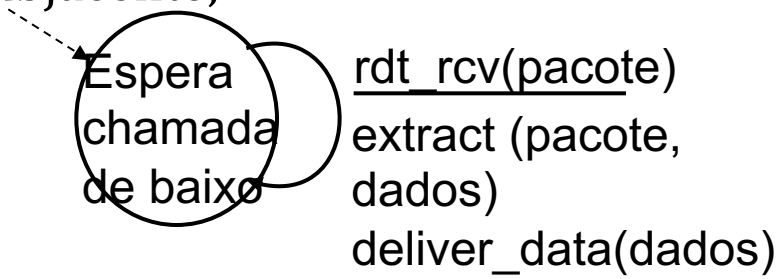


RDT1.0: Transferência Confiável por Canal Confiável

- Canal subjacente perfeitamente confiável:
  - sem erros de bit;
  - sem perda de pacotes;
- FSMs separadas para remetente e destinatário:
  - remetente envia dados para canal subjacente;
  - destinatário lê dados do canal subjacente;



remetente



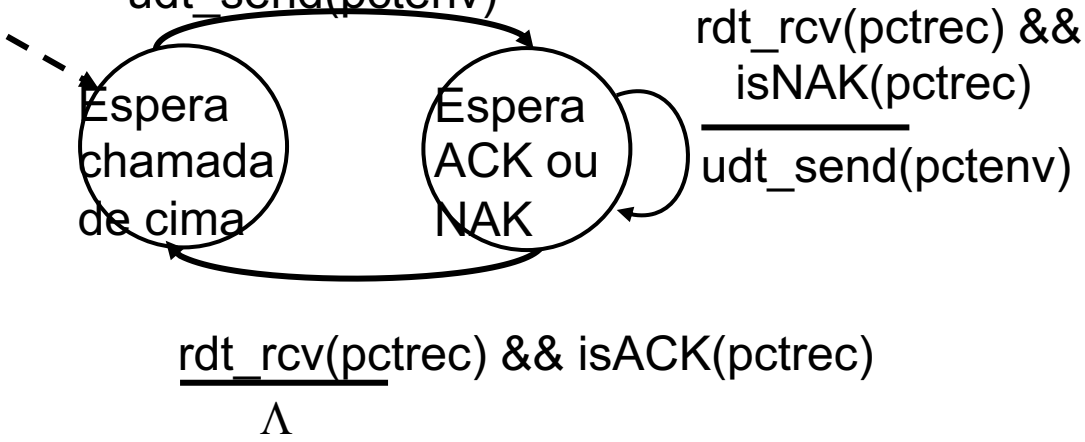
destinatário

RDT2.0: Canal com Erros de Bit

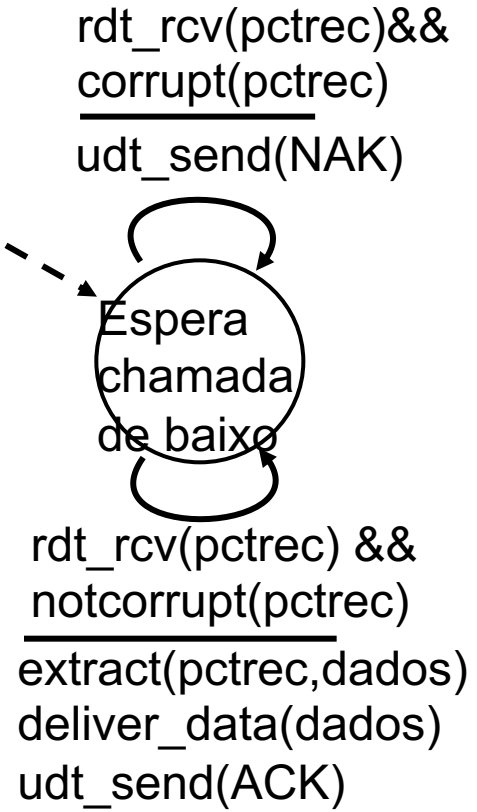
- Canal subjacente pode inverter bits no pacote:
  - soma de verificação para detectar erros de bit;
- Como recuperar-se dos erros?
  - *reconhecimentos (ACKs):* destinatário diz explicitamente ao remetente que o pacote foi recebido OK;
  - *reconhecimentos negativas (NAKs):* destinatário diz explicitamente 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: mensagens de controle (ACK,NAK) destinatário->remetente;

RDT2.0: Especificação da FSM

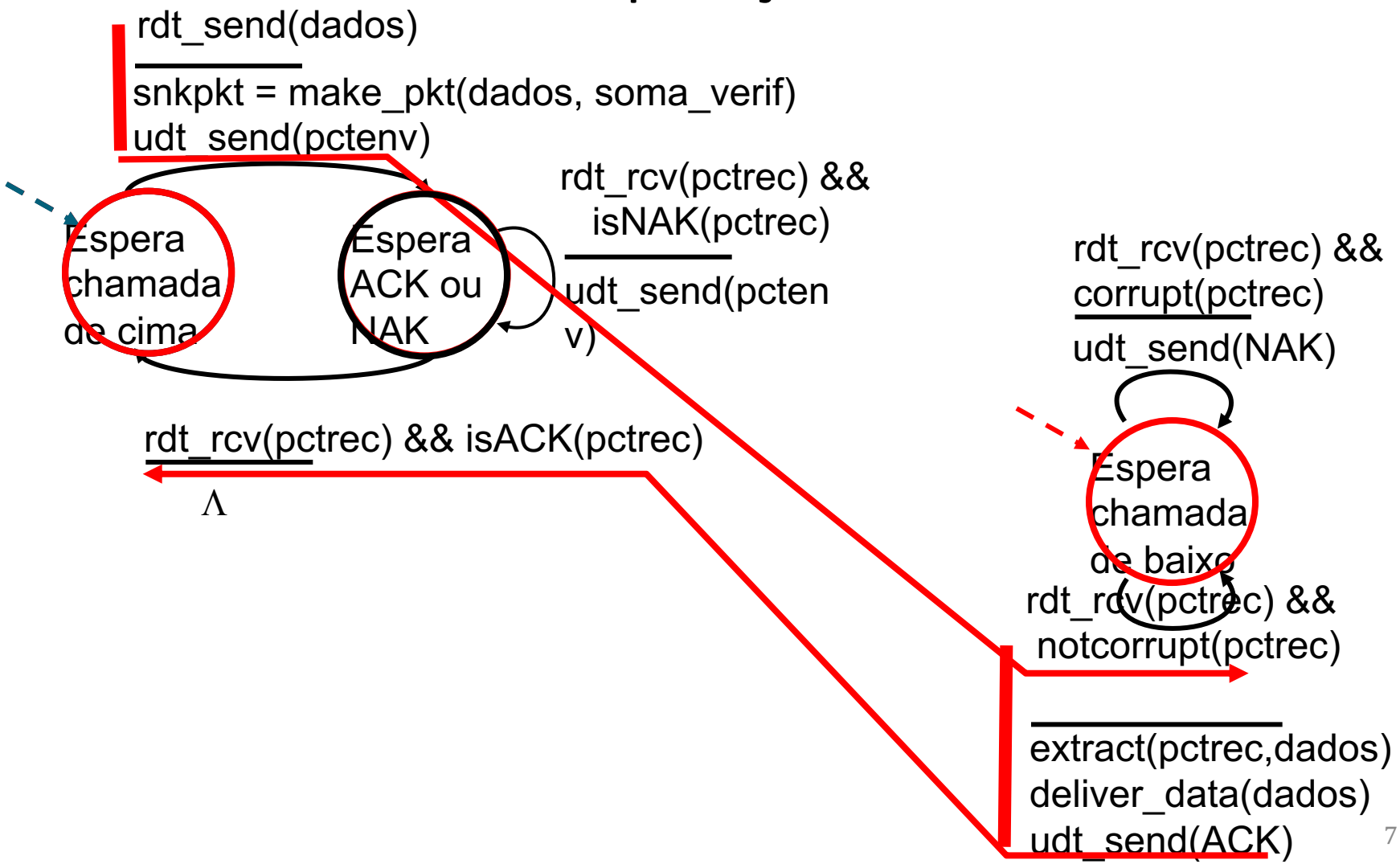
remetente



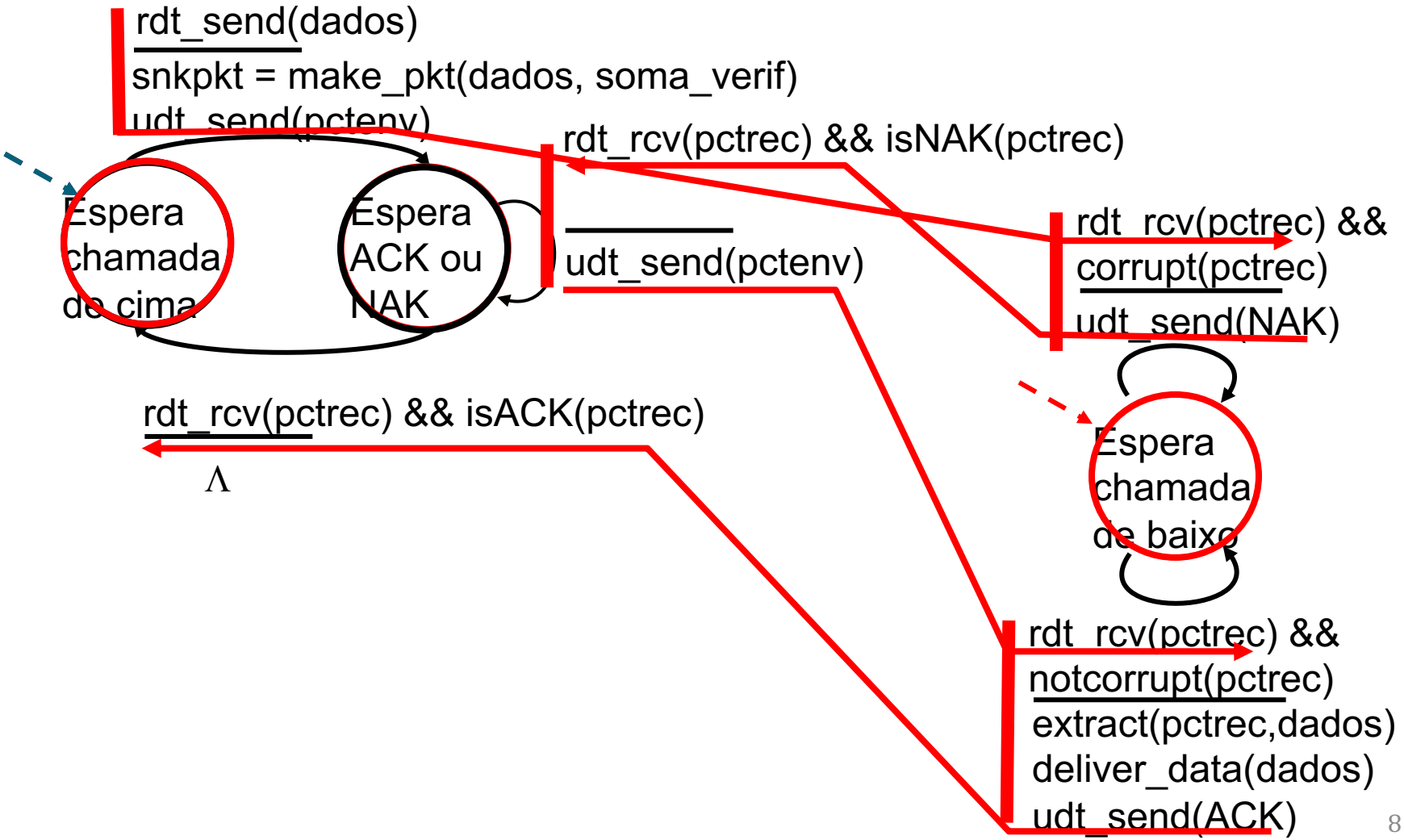
destinatário



RDT2.0: Operação sem Erros



RDT2.0: Cenário de Erro



RDT2.0 : Falha Fatal!

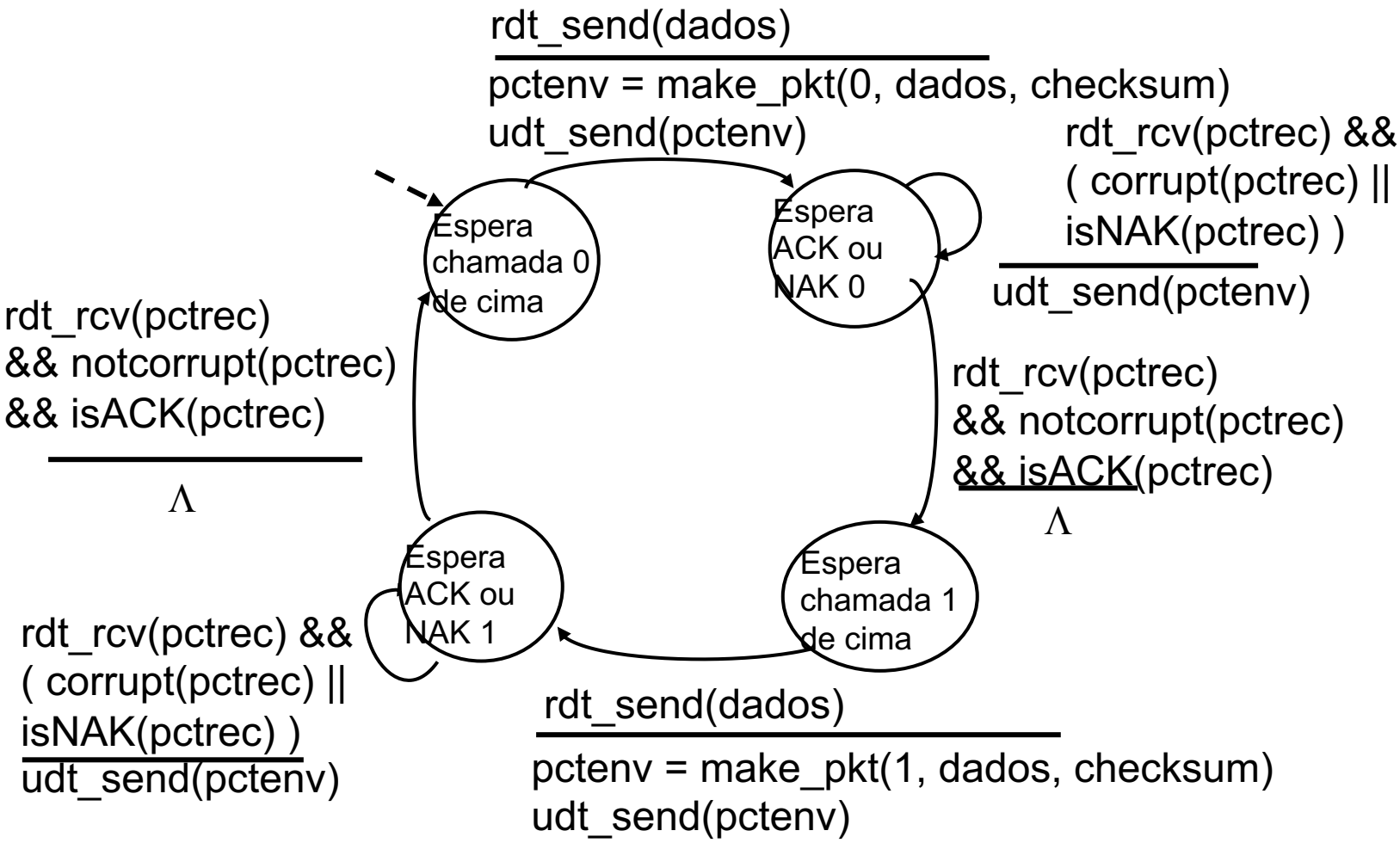
O que acontece se ACK/NAK for corrompido?

- Remetente não sabe o que aconteceu no destinatário;
- Não pode simplesmente retransmitir: possível duplicação;

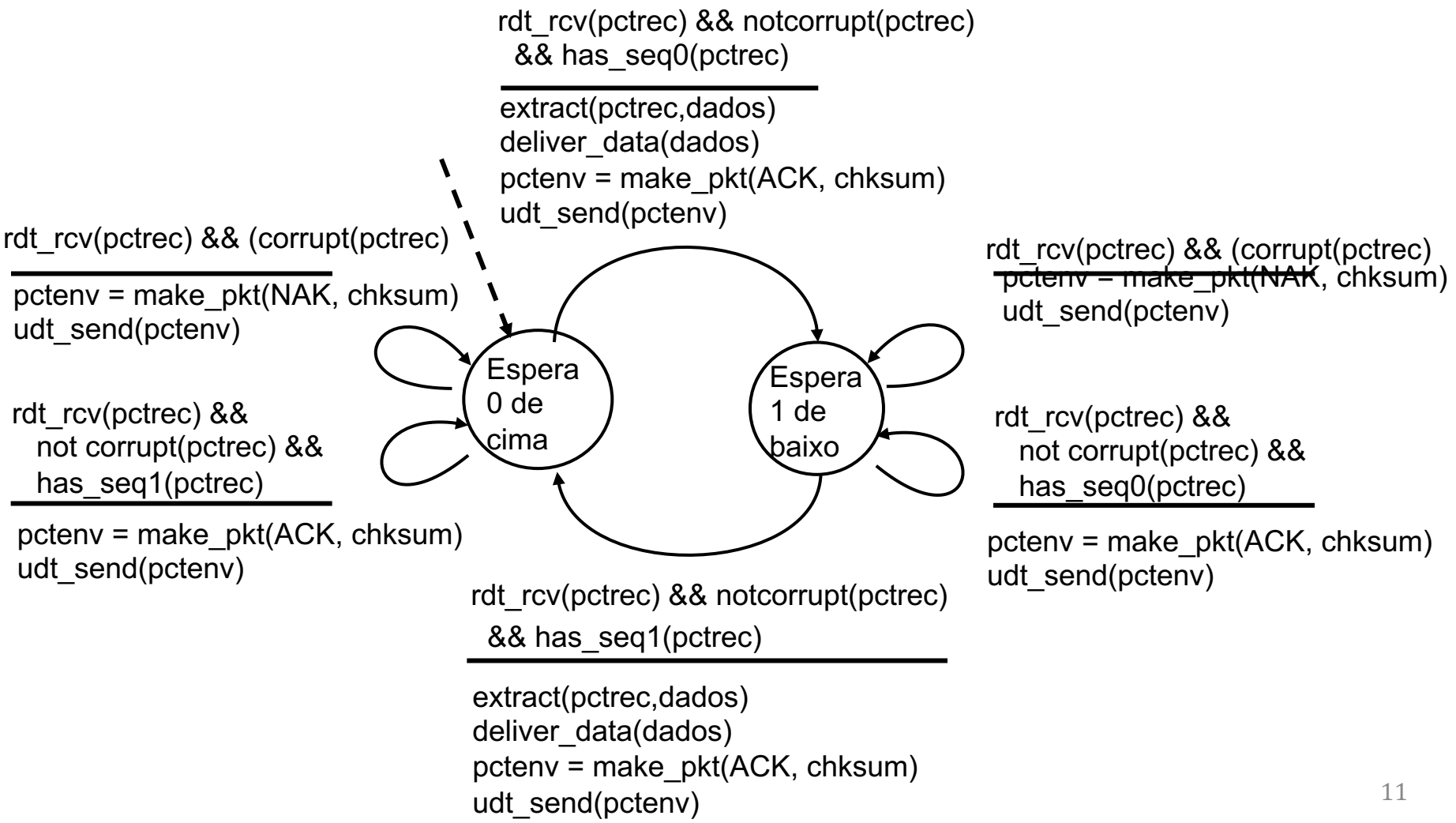
Tratando de duplicatas:

- Remetente retransmite pacote atual se ACK/NAK corrompido;
- Remetente acrescenta *número de sequência* a cada pacote;
- Destinatário descarta pacote duplicado;

RDT2.1: Remetente trata de ACK/NAKs Corrompidos



RDT2.1


rdt_rcv(pctrec) && notcorrupt(pctrec)
&& has_seq0(pctrec)
extract(pctrec,dados)
deliver_data(dados)
pctenv = make_pkt(ACK, chksum)
udt_send(pctenv)
rdt_rcv(pctrec) && (corrupt(pctrec)
pctenv = make_pkt(NAK, chksum)
udt_send(pctenv)
rdt_rcv(pctrec) && (corrupt(pctrec)
pctenv = make_pkt(NAK, chksum)
udt_send(pctenv)
Espera
0 de
cima
Espera
1 de
baixo
rdt_rcv(pctrec) &&
not corrupt(pctrec) &&
has_seq1(pctrec)
pctenv = make_pkt(ACK, chksum)
udt_send(pctenv)
rdt_rcv(pctrec) &&
not corrupt(pctrec) &&
has_seq0(pctrec)
pctenv = make_pkt(ACK, chksum)
udt_send(pctenv)
rdt_rcv(pctrec) && notcorrupt(pctrec)
&& has_seq1(pctrec)
extract(pctrec,dados)
deliver_data(dados)
pctenv = make_pkt(ACK, chksum)
udt_send(pctenv)

RDT2.1

remetente:

- # seq acrescentado ao pacote;
- 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

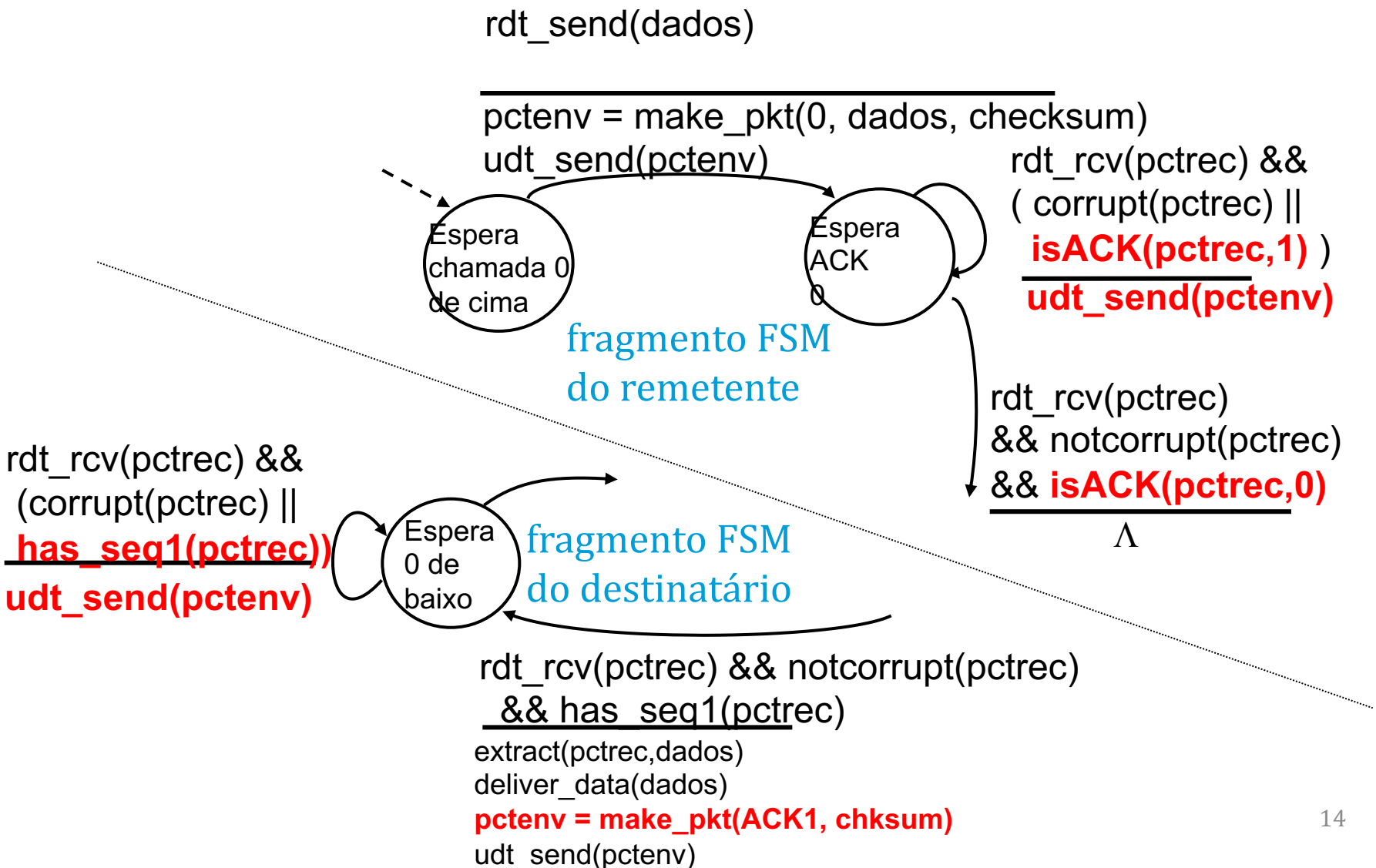
destinatário:

- 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;

RDT 2.2: 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;*

RDT 2.2: Fragmentos do Remetente, Destinatário



RDT 3.0: Canais com Erros *e* Perda

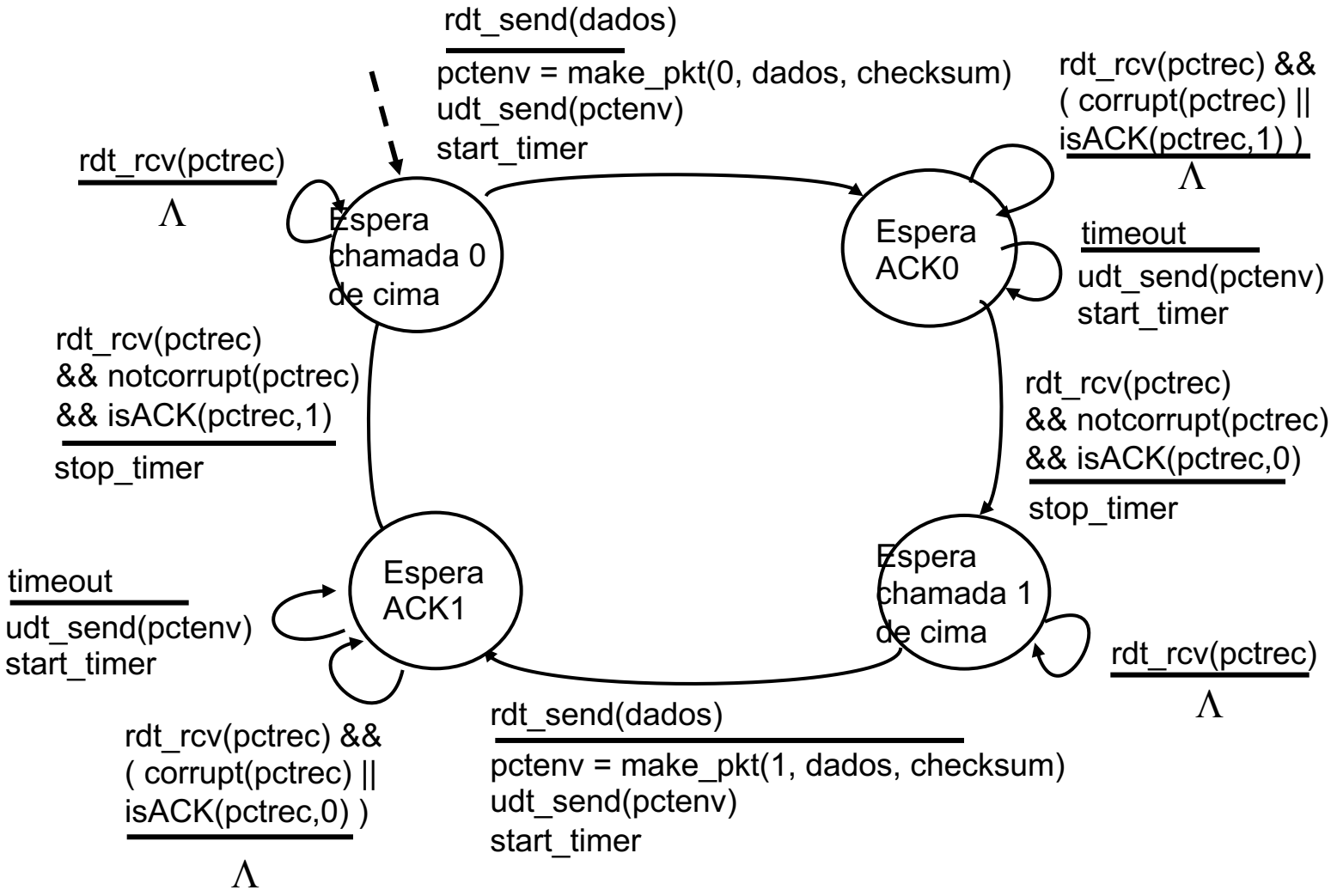
nova suposição: canal subjacente também pode perder pacotes (dados ou ACKs)

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

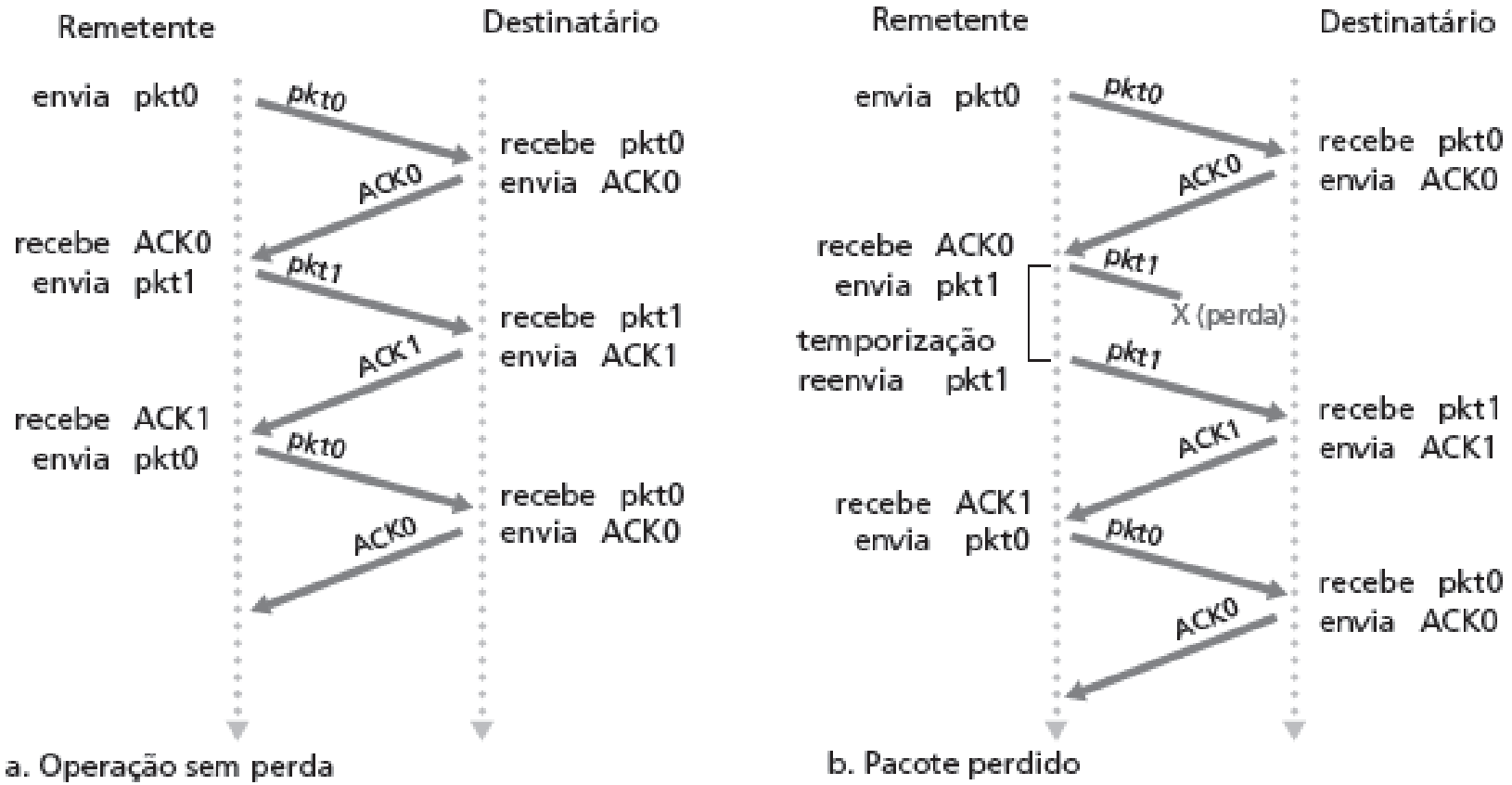
técnica: remetente espera quantidade "razoável" de tempo por ACK

- retransmite se não chegar ACK nesse tempo
- se pacote (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;

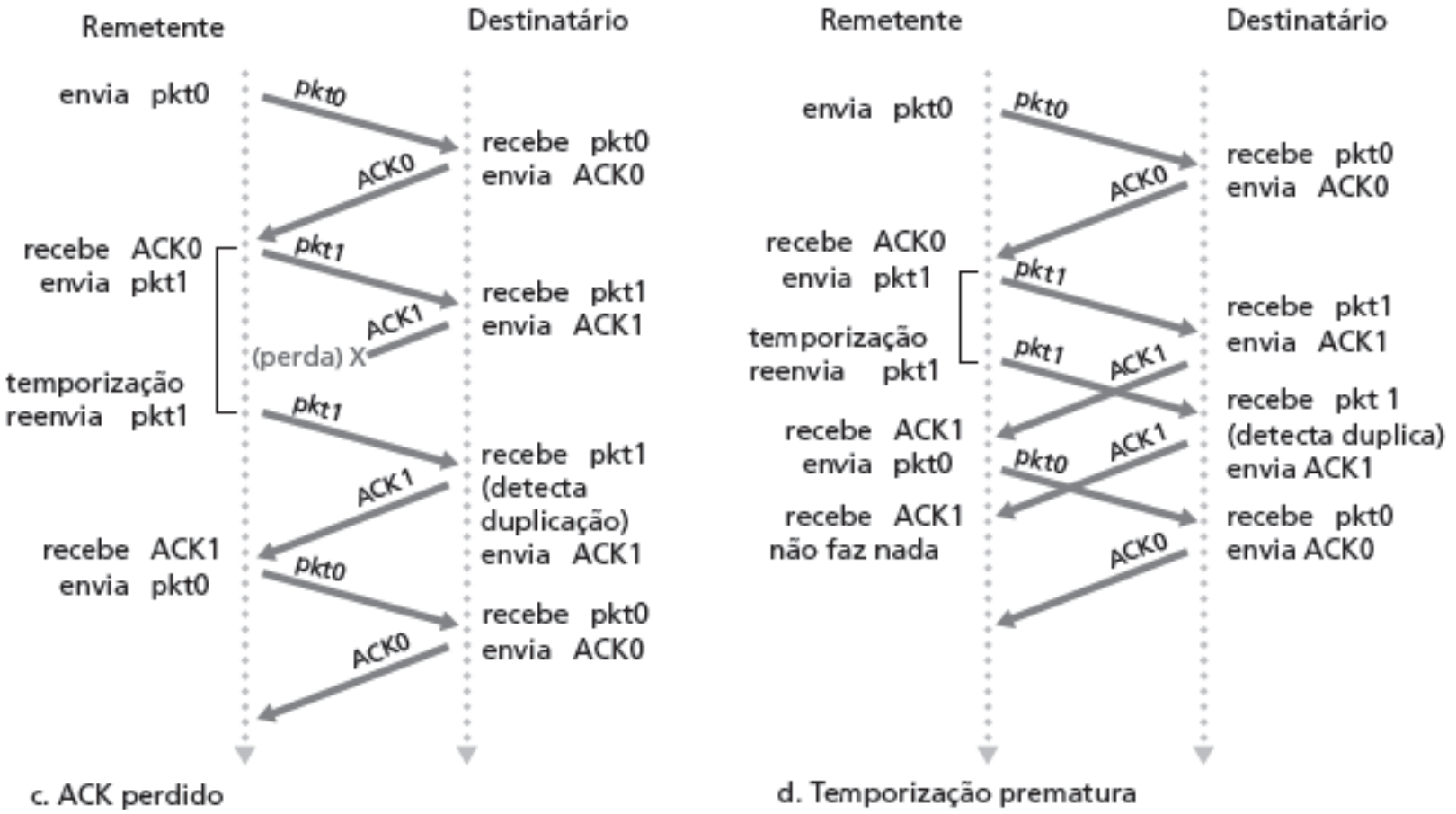
Remetente RDT 3.0



RDT3.0 em ação



RDT3.0 em ação



Desempenho do RDT3.0

- RDT 3.0 funciona, mas com desempenho ruim;
- ex.: enlace 1 Gbps, 15 ms atraso propriedade, pacote 8000 bits:

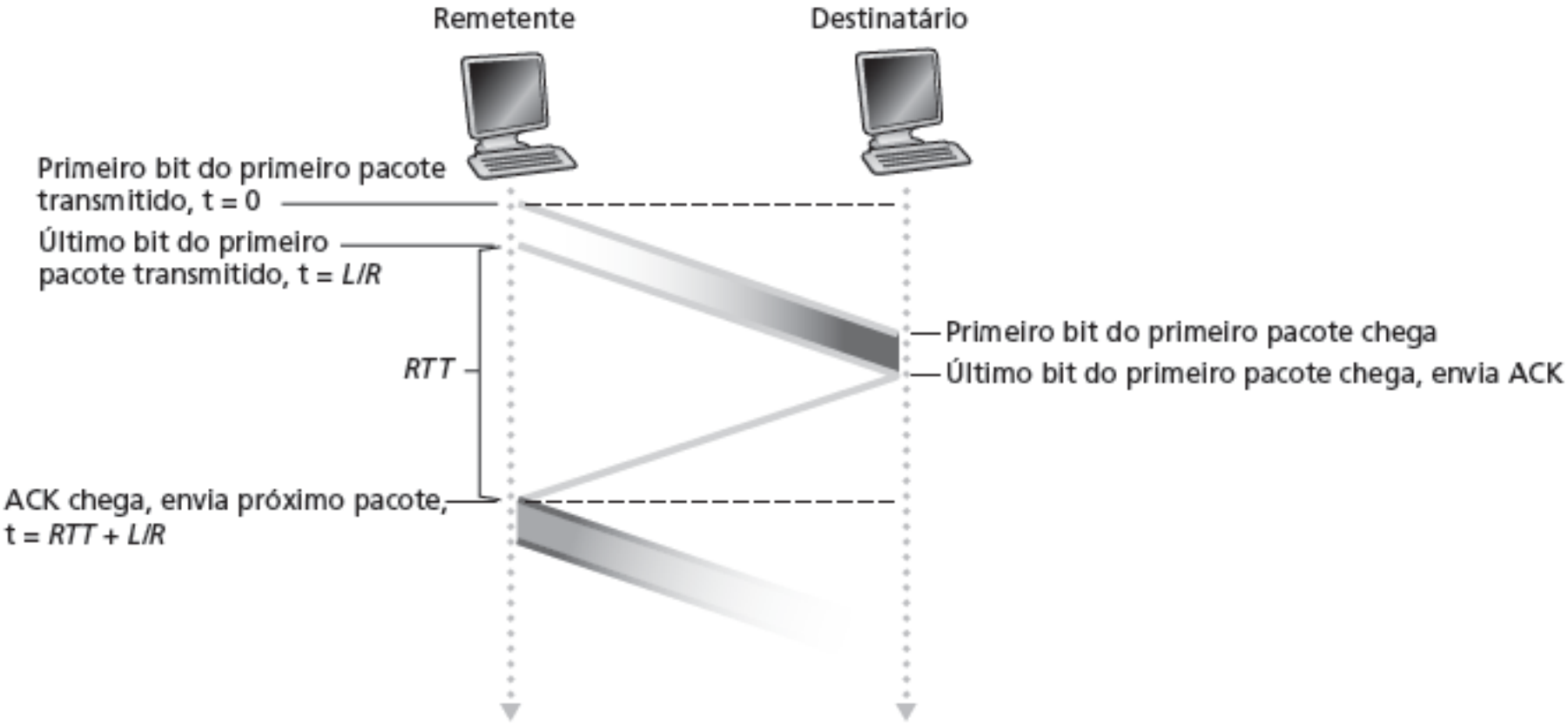
$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\,\text{bps}} = 8\,microssegundos$$

  - U_{remet}: utilização – fração do tempo remetente ocupado enviando

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

  - Pacote 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

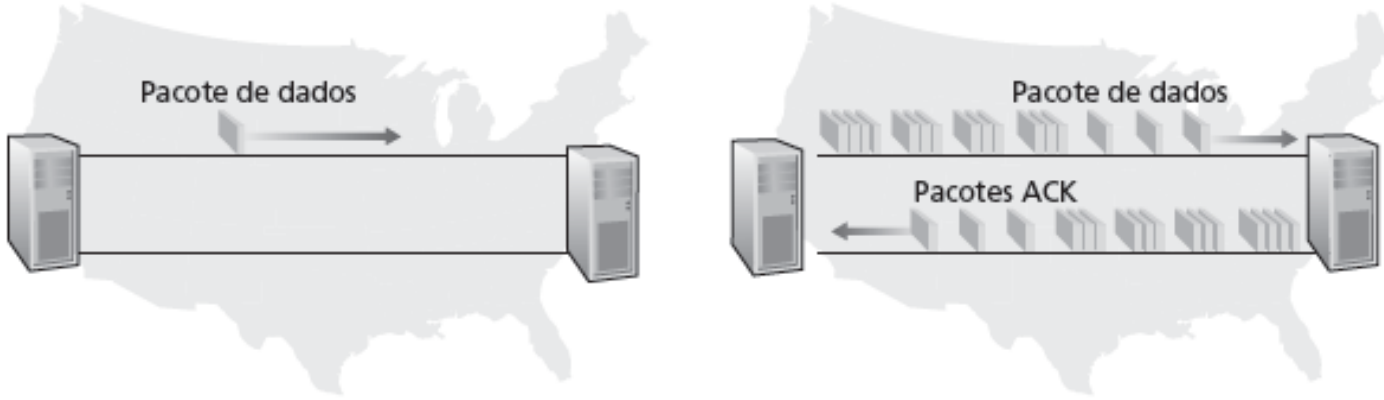


Protocolos com paralelismo

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

–

–

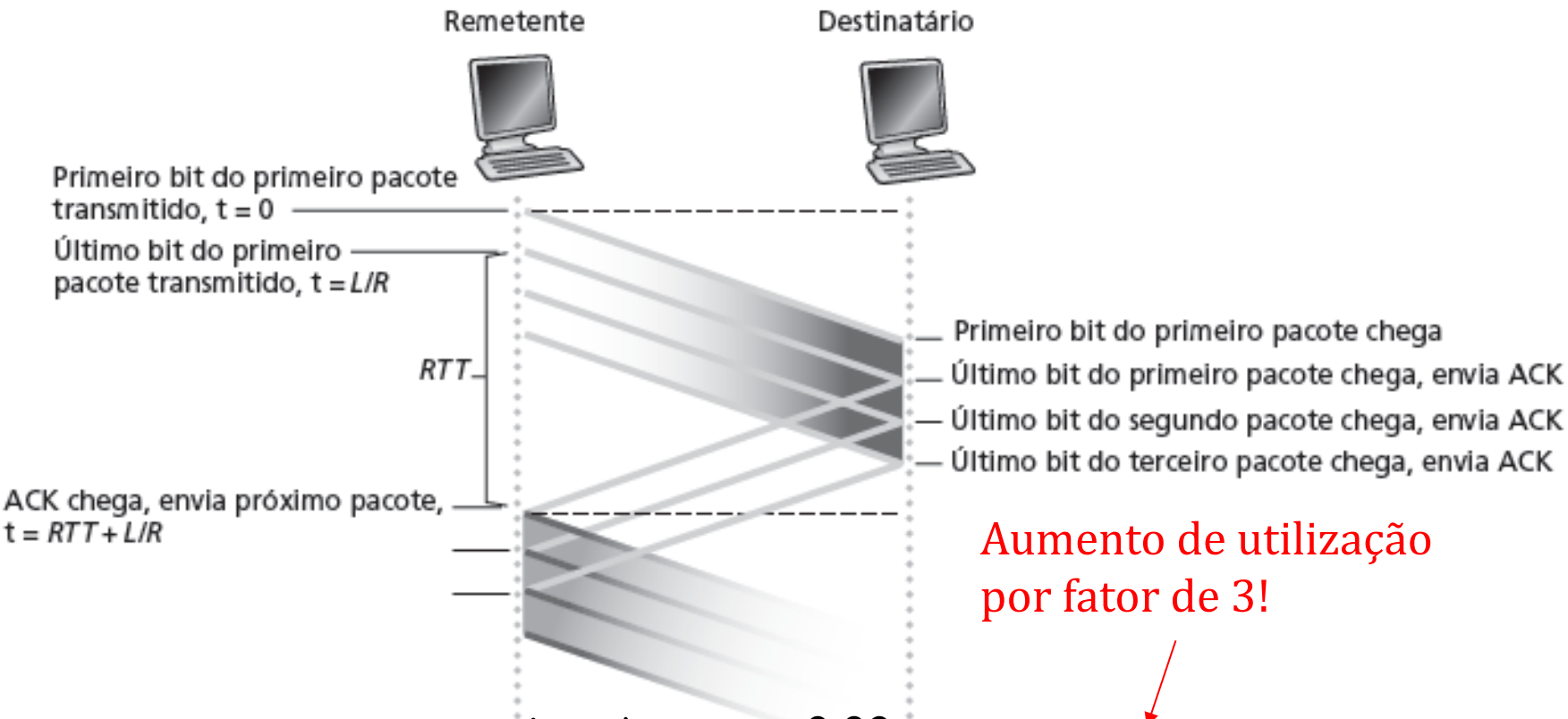


a. Um protocolo pare e espere em operação

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 pacote ACK se houver uma lacuna;
- *remetente:* tem temporizador para pacote sem ACK mais antigo;
  - se o temporizador expirar: retransmite todos os pacotes sem ACK;

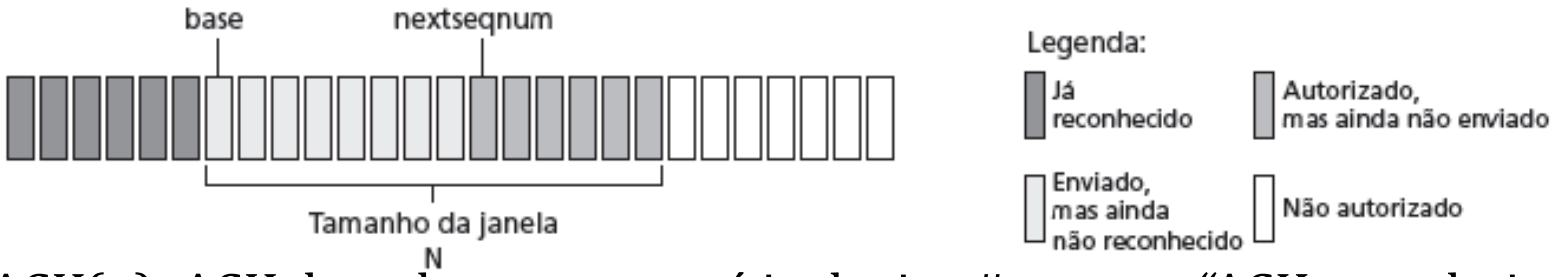
Repetição seletiva: visão geral

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

Go-Back-N

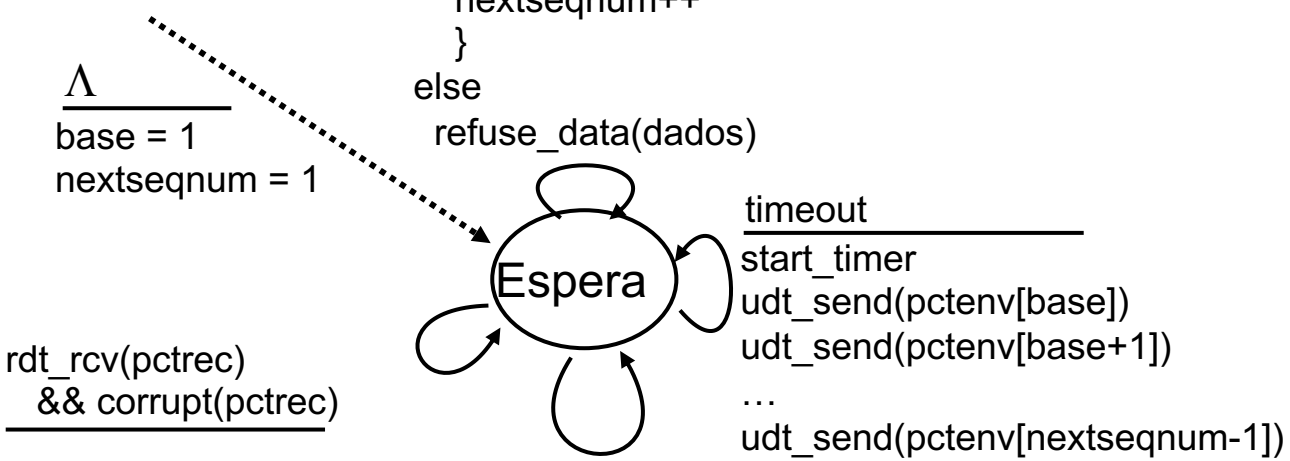
remetente:

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

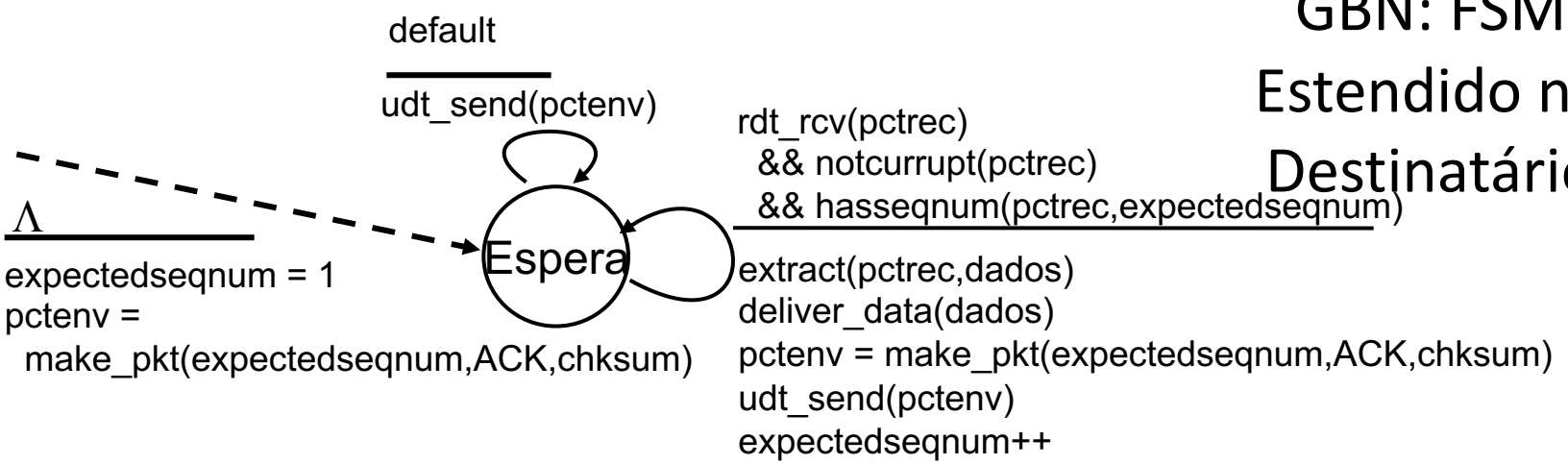


- ACK(n): ACK de todos pacotes até inclusive # seq. n – "ACK cumulativo"
  - pode receber ACKs duplicados (ver destinatário);
- temporizador para cada pacote no ar;
- *timeout(n):* retransmite pacote n e todos pacotes com # seq. mais alto na janela;

GBN: FSM estendido no remetente



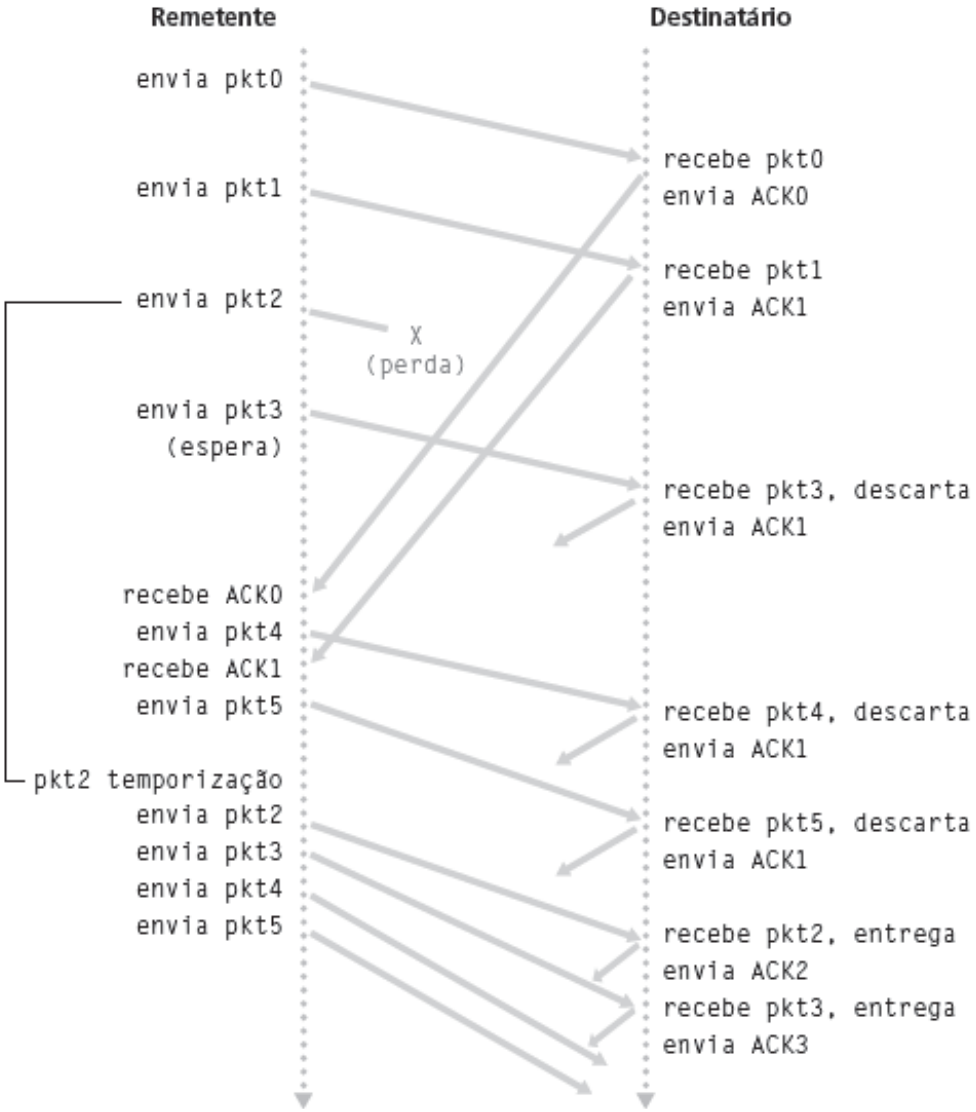
GBN: FSM Estendido no Destinatário



- Apenas ACK: sempre envia ACK para pacote 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;

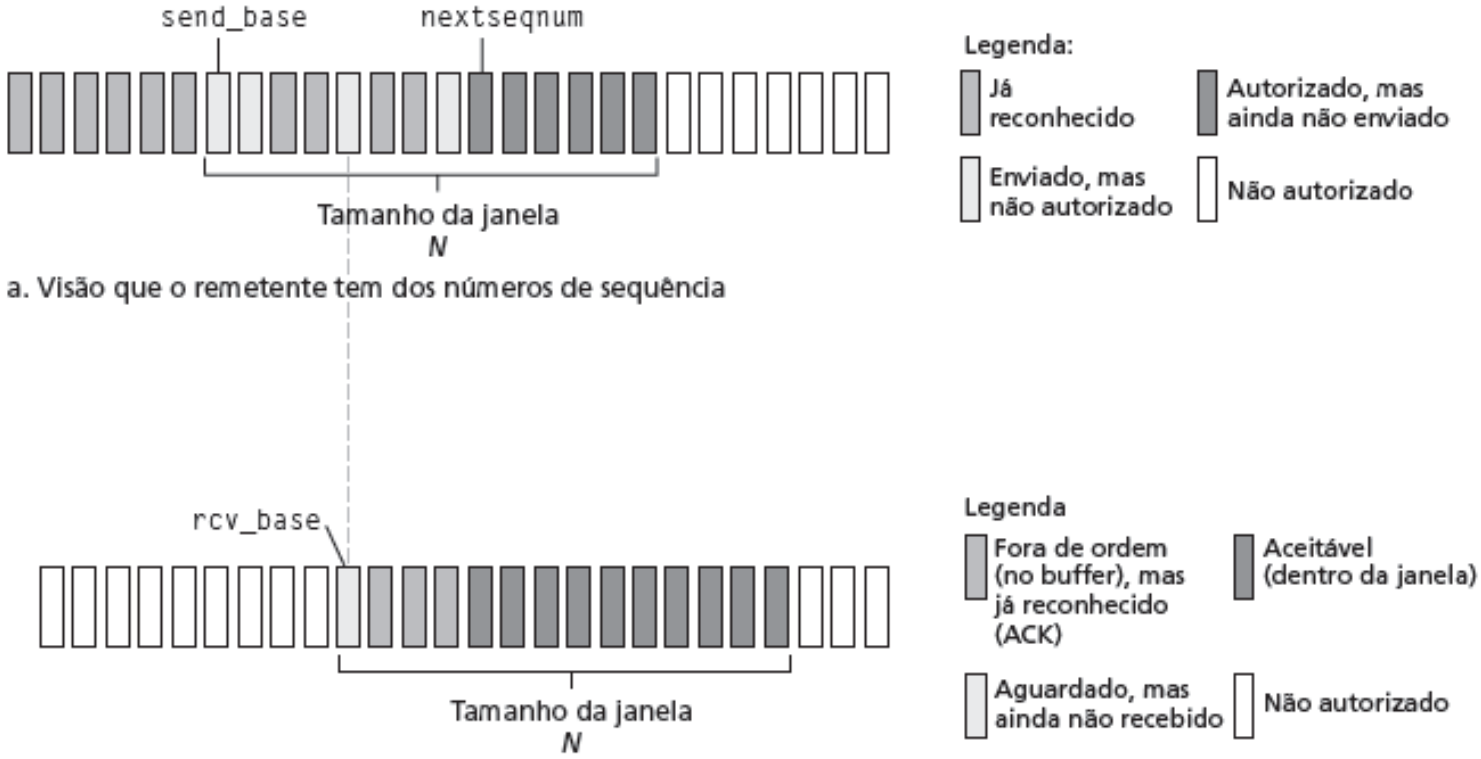
GBN em operação



Repetição seletiva

- Destinatário reconhece *individualmente* todos os pacotes recebidos de modo correto:
  - mantém pacotes em buffer, se for preciso, para eventual remessa em ordem para a camada superior;
- Remetente só reenvia pacotes para os quais o ACK não foi recebido;
  - temporizador no remetente para cada pacote sem ACK;
- Janela do remetente:
  - N # seq. Consecutivos;
  - Novamente limita #s seq. de pacotes enviados, sem ACK;

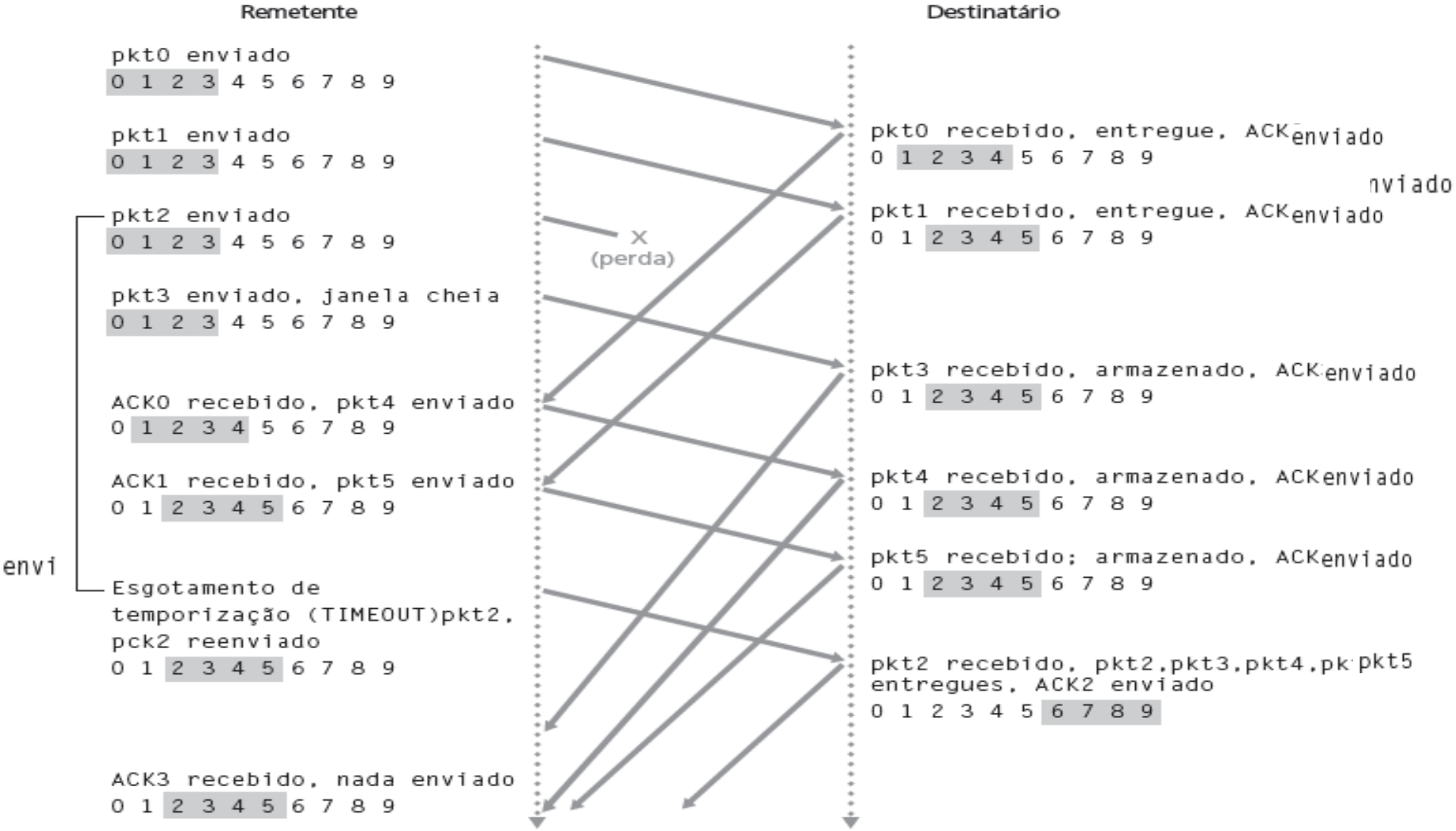
Repetição seletiva: janelas de remetente, destinatário



a. Visão que o remetente tem dos números de sequência

b. Visão que o destinatário tem dos números de sequência

Repetição seletiva em operação



Link: Applet GBN

- https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/go-back-n-protocol/index.html

Link: Applet SR

- https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/selective-repeat-protocol/index.html