

PTC 3360

2.2 Camada de transporte: princípios da transferência confiável de dados – Parte I

(Kurose, Seção 3.4)

Agosto 2025

Capítulo 2 - Conteúdo

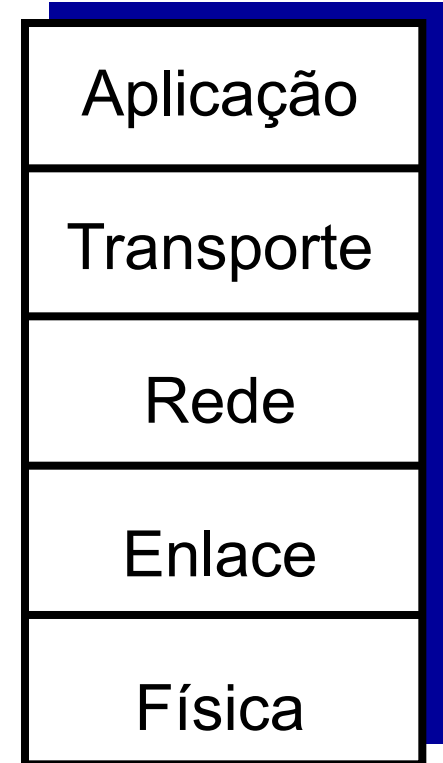
2.1 A camada de aplicação

2.2 A camada de transporte: Princípios da transferência confiável de dados

2.3 A camada de rede

Lembrando... Pilha de protocolos Internet

- ❖ **Aplicação:** contendo aplicativos de rede que geram **mensagens**
 - HTTP, SMTP, Apple Airplay, DNS, ...
- ❖ **Transporte:** transferência de **segmentos** processo-processo
 - **TCP**, UDP, QUIC
- ❖ **Rede:** roteamento de **datagramas** da fonte ao destino
 - IP, protocolos de roteamento
- ❖ **Enlace:** transferência de **quadros** entre **elementos vizinhos** na rede
 - Ethernet, WiFi, DOCSIS, ...
- ❖ **Física:** transmissão física dos **bits**; depende do meio de transmissão

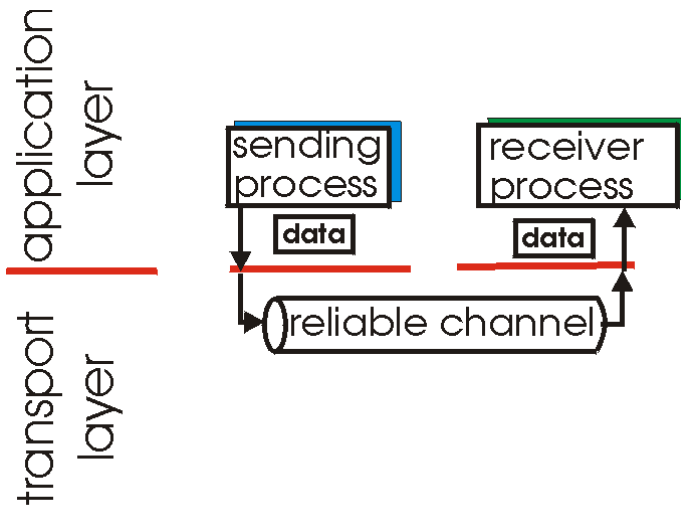


RDT – Reliable Data Transfer

- ❖ Um dos aspectos mais importantes das redes de comunicações modernas é a *transferência confiável de dados* (RDT – *reliable data transfer*).
- ❖ Entre fonte e destino, pacotes passam por diversos enlaces diferentes; neste caminho, podem acontecer muitos problemas. Eles podem
 - ser descartados em cada um dos roteadores no meio da rota;
 - ser corrompidos;
 - percorrer rotas diferentes, tendo atrasos diferentes, chegando inclusive fora de ordem.
- ❖ Mesmo assim, muitas aplicações são viáveis apenas se há garantia de que todos os pacotes sejam entregues e em ordem: comércio eletrônico, e-mail, transferência de arquivos de dados, etc.
- ❖ Nessa aula, vamos estudar técnicas que são utilizadas por diversos protocolos para implementar a RDT.
- ❖ Em seguida, como exemplo, vamos ver como o **TCP** implementa essas técnicas.

Princípios da transferência confiável de dados

- ❖ RDT é usada nas camadas de aplicação, transporte e enlace
 - Tópico de importância central na área de redes!

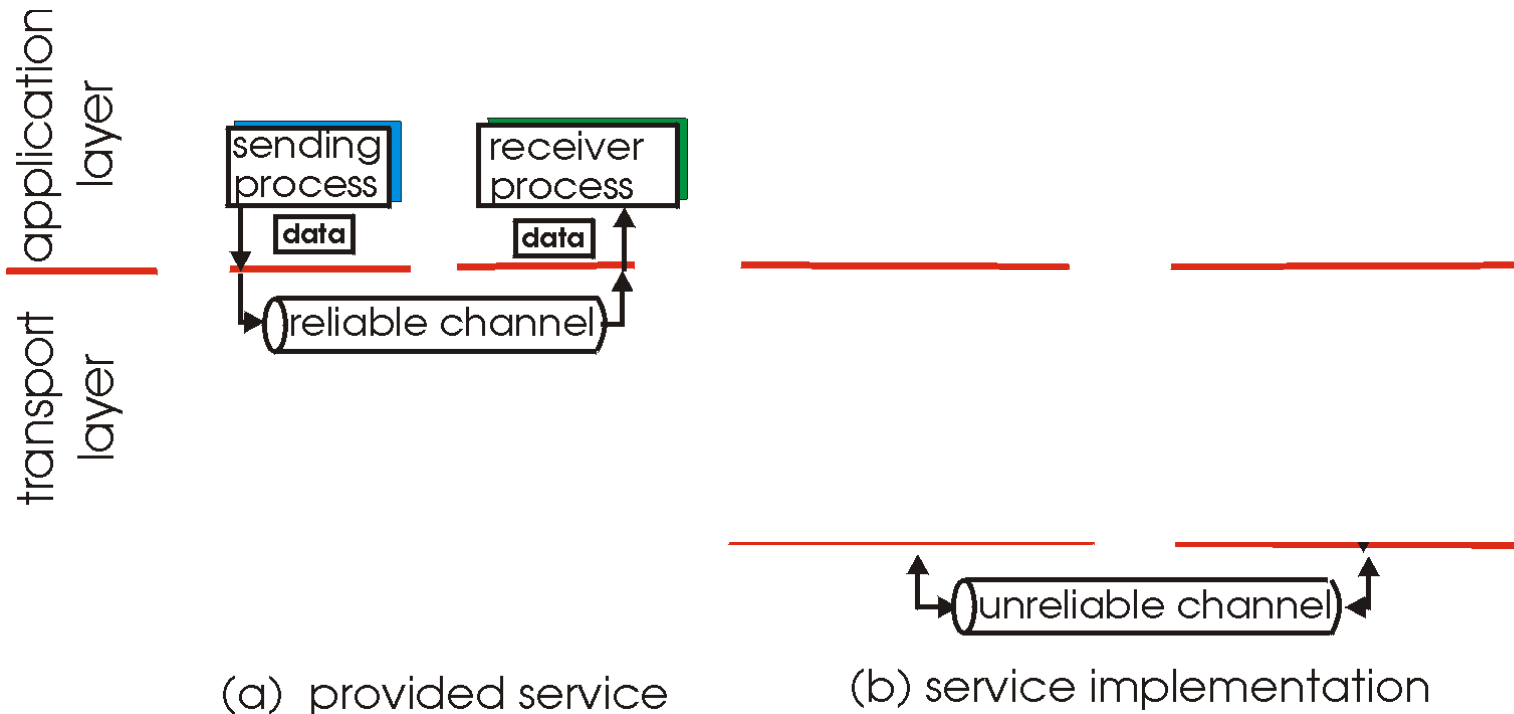


(a) provided service

- ❖ características do canal não confiável determinarão complexidade de um protocolo que implementa RDT

Princípios da transferência confiável de dados

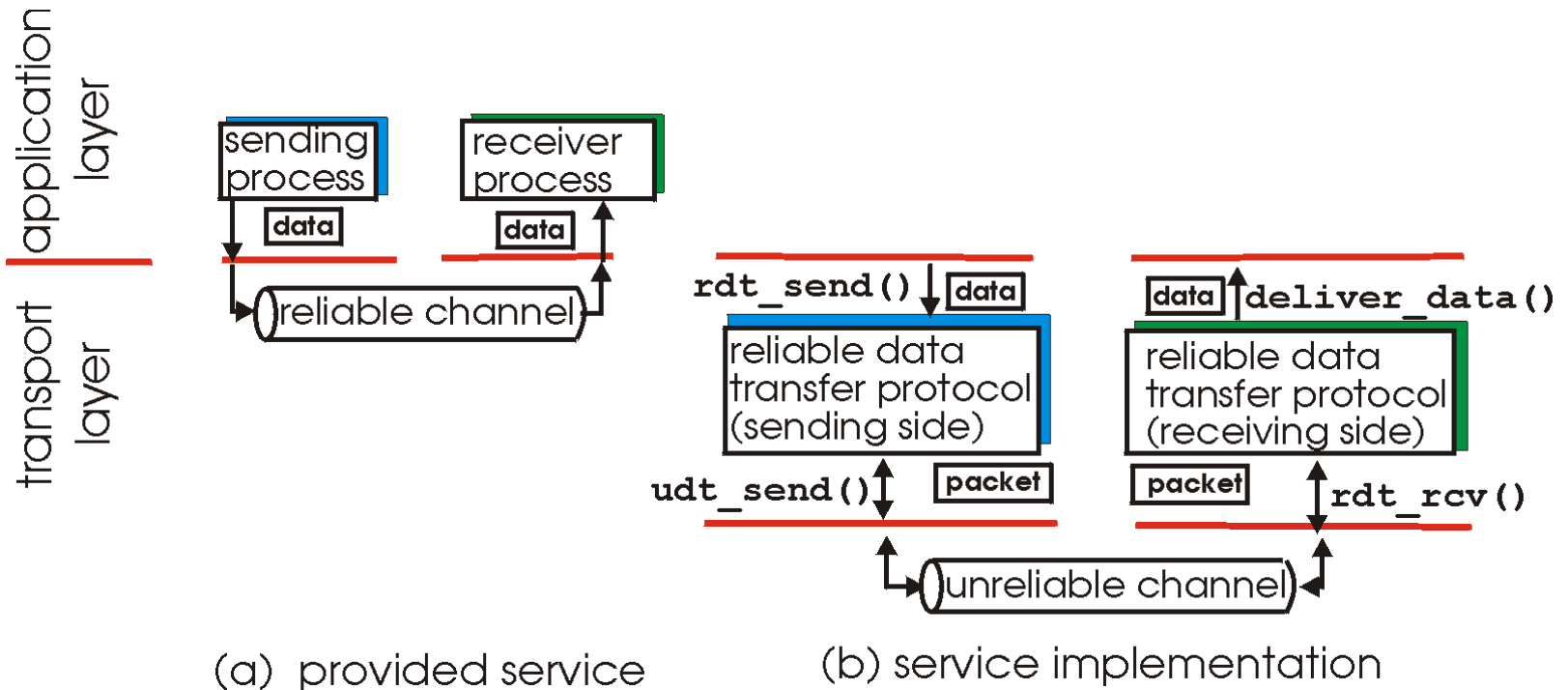
- ❖ RDT usada nas camadas de aplicação, transporte e enlace
 - Tópico de importância central na área de redes!



- ❖ características do canal não confiável determinarão complexidade de um protocolo que implementa RDT

Princípios da transferência confiável de dados

- ❖ RDT usada nas camadas de aplicação, transporte e enlace
 - Tópico de importância central na área de redes!

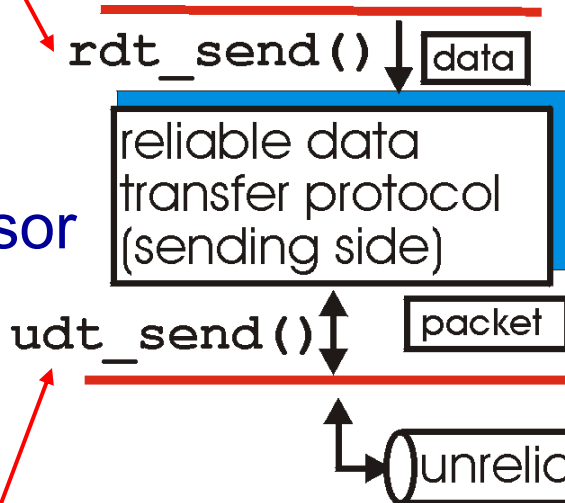


- ❖ Características do canal não confiável determinarão, em parte, complexidade de um protocolo que implementa RDT

Transferência confiável de dados: rotinas

rdt_send() : chamada de cima, por aplicação. São passados dados a serem entregues à camada superior do destinatário (*reliable data transfer*)

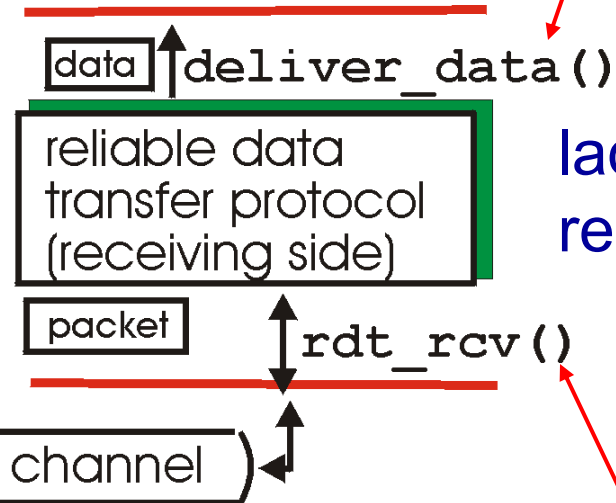
lado
transmissor



udt_send() : chamada pelo rdt, para transferir pacote sobre canal não confiável ao destinatário (*unreliable data transfer*)

deliver_data() : chamada pelo **rdt** para entregar dados para camada superior

lado
receptor

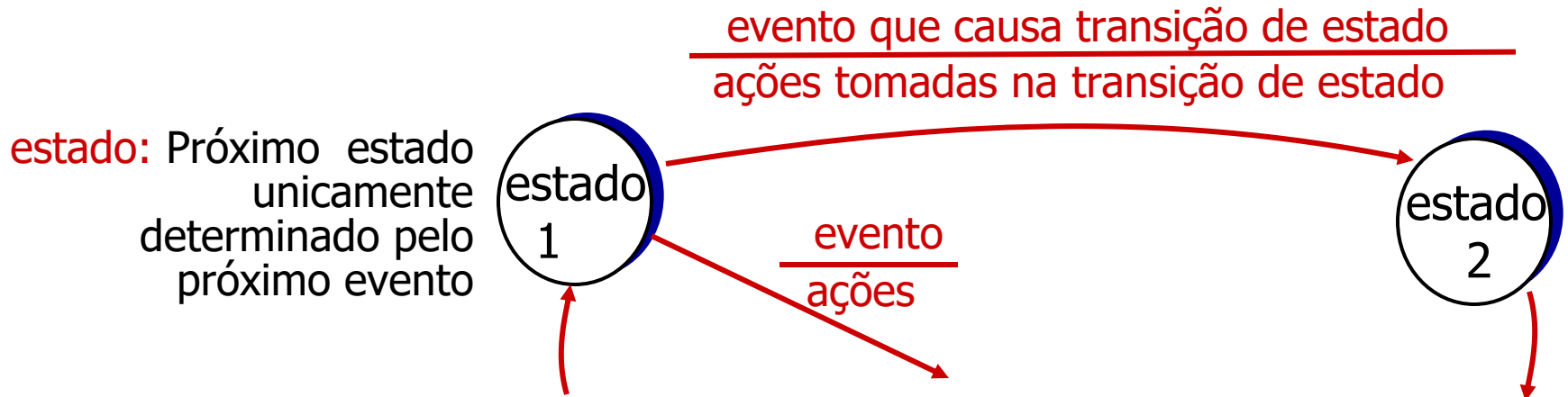


rdt_rcv() : chamada quando pacote chega no lado receptor do canal

Transferência confiável de dados: começando

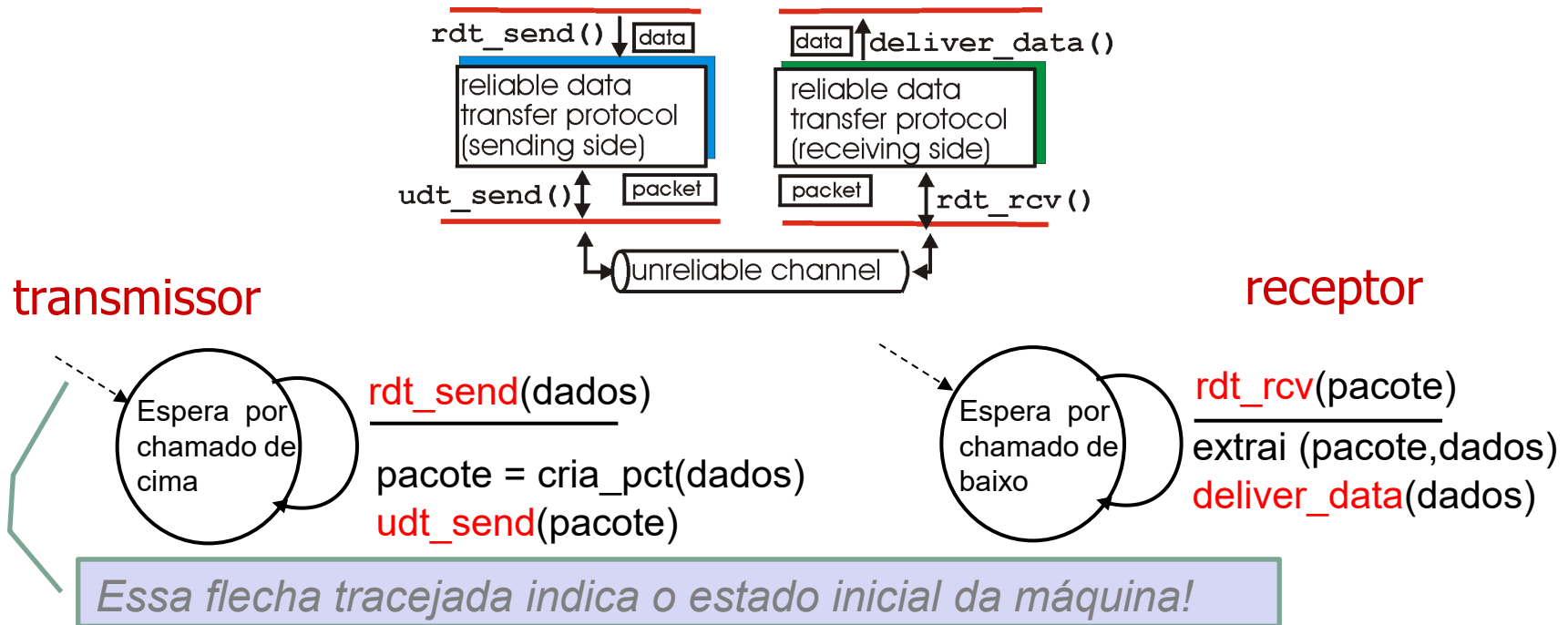
Vamos:

- ❖ Desenvolver lados transmissor e receptor do protocolo RDT de forma incremental (**versões didáticas**);
- ❖ Considerar apenas *transferência unidirecional de dados*;
 - Mas note que as informações de controle fluirão em ambas direções.
- ❖ Usar máquinas de estados para especificar algoritmos no transmissor e receptor



rdt1.0: RDT sobre canal confiável

- ❖ Canal subjacente perfeitamente confiável
 - Não há erros em bits, nem perdas de pacotes
- ❖ Separar máquinas de estados para transmissor e receptor:
 - Remetente envia dados para o canal subjacente
 - Destinatário lê dados do canal subjacente



rdt2.0: Canal com erros em bits

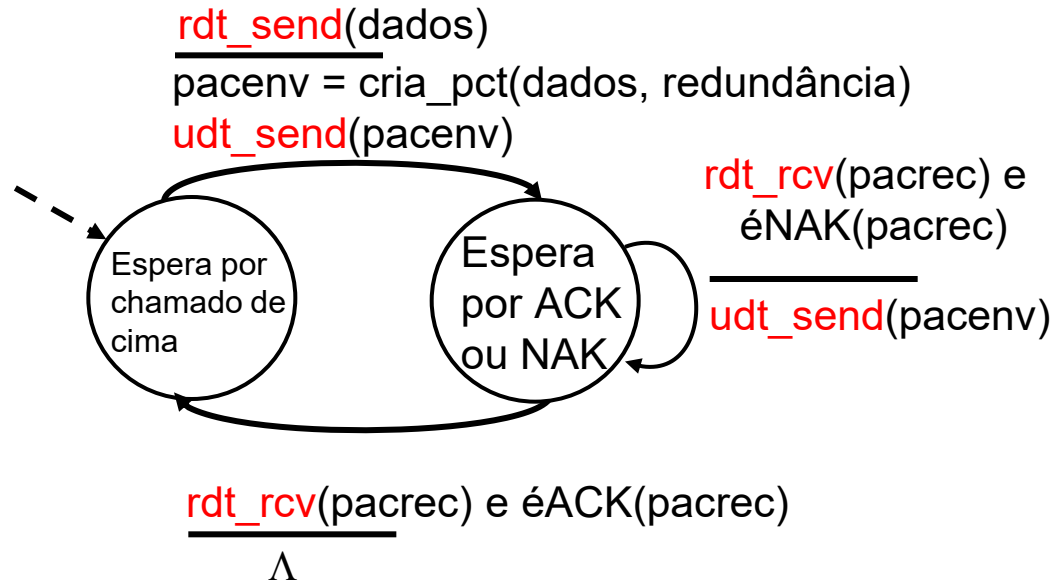
- ❖ Canal subjacente **pode corromper bits em pacote**, mas não perde ou muda ordem de pacotes.
 - Podemos usar **códigos** (por exemplo, bits de paridade) para detectar erros em bits. Vamos assumir que esses códigos de fato detectam os erros.
 - **A questão: como se recuperar de erros?**

Como humanos se recuperam de “erros” durante conversação?

rdt2.0: canal com erros em bits

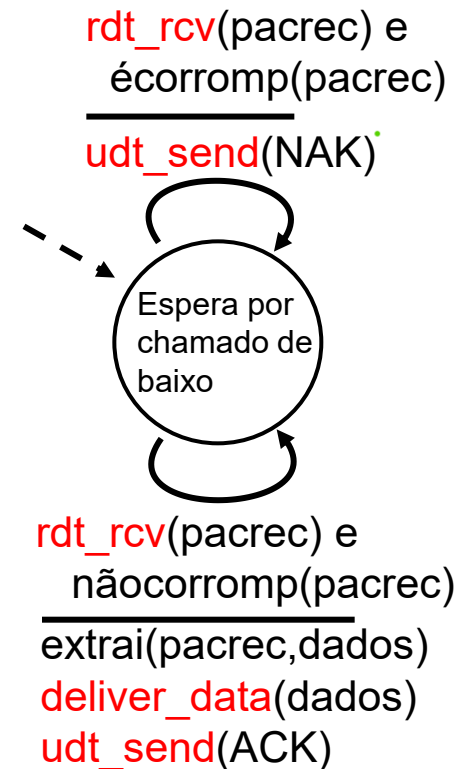
- ❖ canal subjacente **pode corromper bits em pacote**, mas não perde ou muda ordem de pacotes
 - podemos usar **códigos** (por exemplo, bits de paridade) para detectar erros em bits. Vamos assumir que esses códigos de fato detectam os erros.
 - **a questão: como se recuperar de erros?**
 - **Positive acknowledgements (ACKs)**: receptor explicitamente conta ao transmissor que pacote recebido não possui erros
 - **Negative acknowledgements (NAKs)**: receptor explicitamente conta ao transmissor que pacote tem erros
 - Transmissor retransmite pacote quando recebe NAK
- ❖ Novos mecanismos em **rdt2.0** (evolução do **rdt1.0**):
 - **Detecção de erros** (necessária inclusão de redundância nos pacotes)
 - **Realimentação**: mensagens de controle (ACK, NAK) do receptor para transmissor
 - **Retransmissão**: pacotes com erro são retransmitidos

rdt2.0: Especificação por máquina de estados



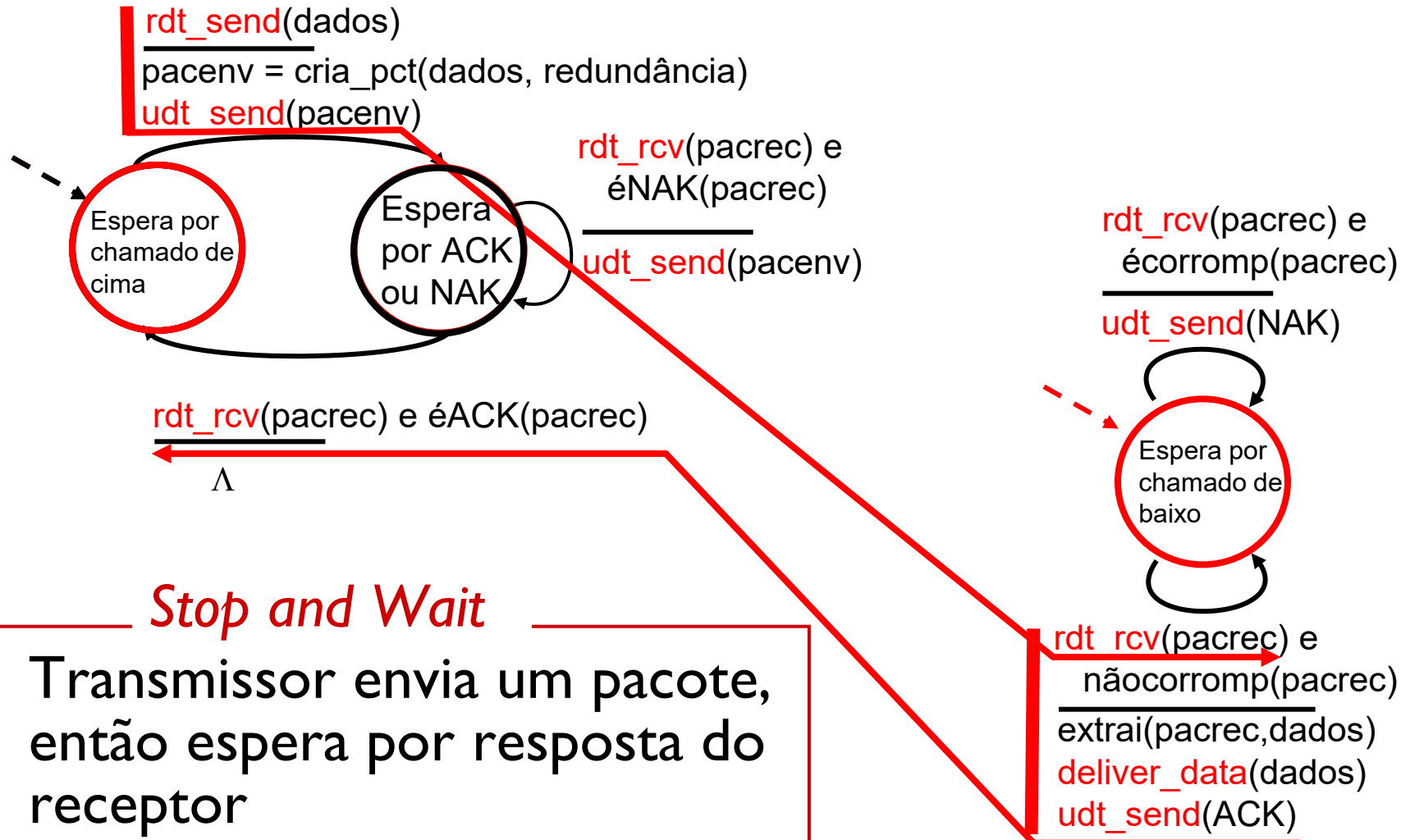
Transmissor

Receptor

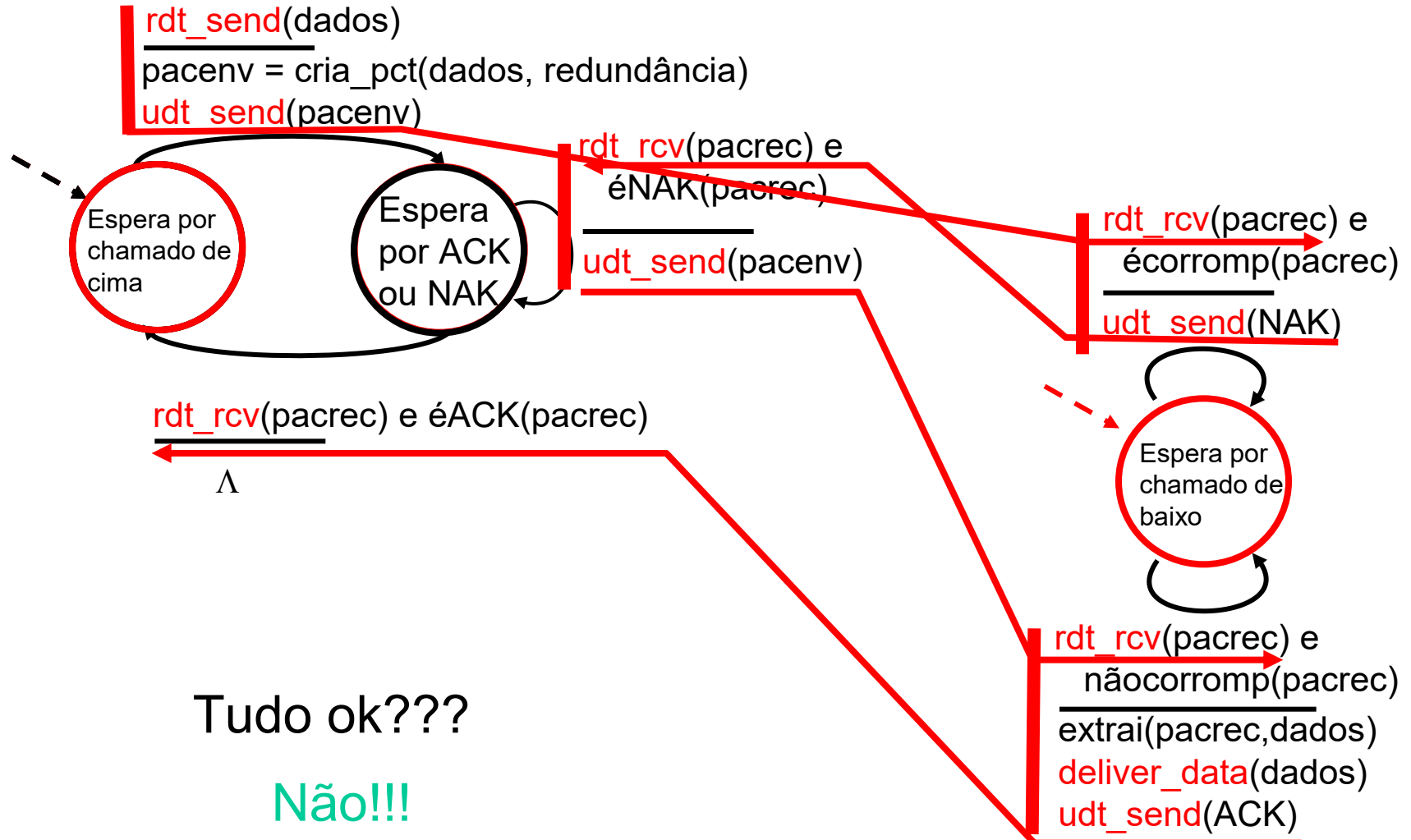


Obs.: O símbolo Λ indica que nenhuma ação é realizada

rdt2.0: Operação sem erros



rdt2.0: Cenário com erro



rdt2.0 tem um defeito fatal!

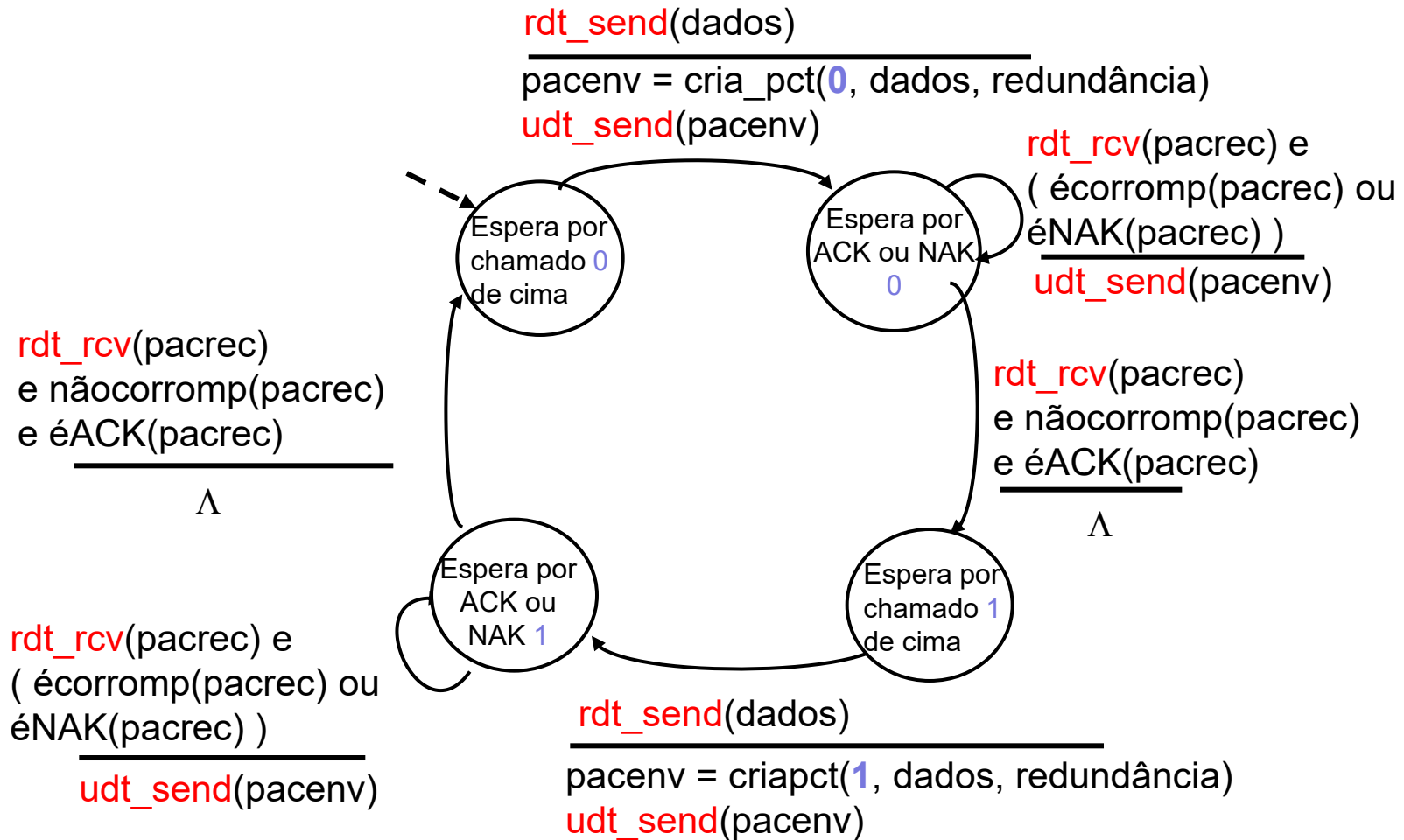
O que acontece se ACK/NAK corrompido?

- ❖ Transmissor não sabe o que aconteceu no receptor!
- ❖ Pode simplesmente retransmitir?
- ❖ **Não! Possível duplicata!**
- ❖ **Como resolver?**

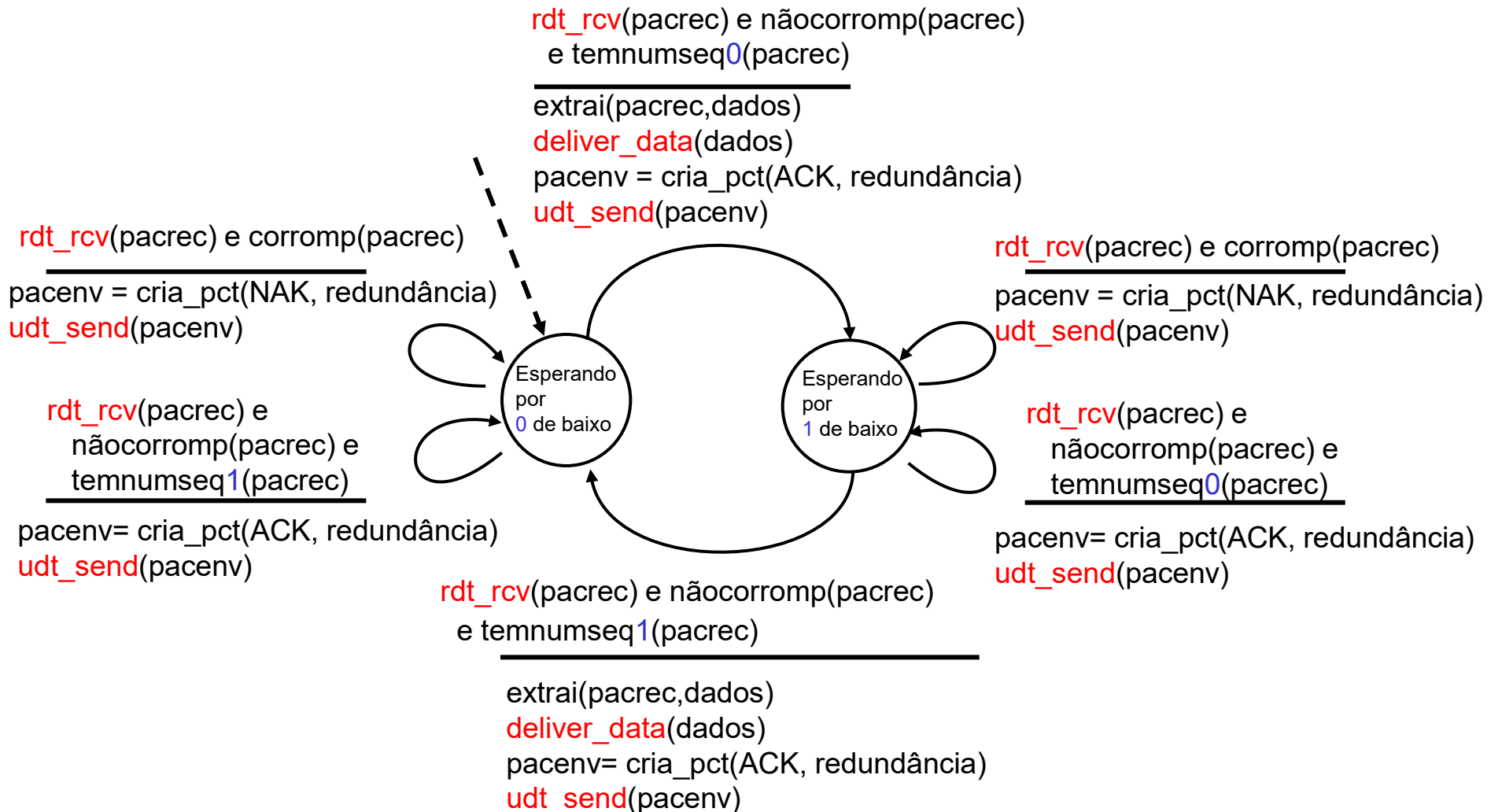
Lidando com duplicatas:

- ❖ Transmissor retransmite pacote corrente se ACK/NAK corrompido
- ❖ Transmissor adiciona *número sequencial* a cada pacote
- ❖ Receptor descarta (não entrega) pacote duplicado

rdt2.1: Transmissor, lidando com ACK/NAKs corrompidos



rdt2.1: Receptor, lidando com ACK/NAKs corrompidos



rdt2.1: Discussão

Transmissor:

- ❖ Número sequencial ($\#seq$) adicionado ao pacote.
- ❖ 2 números sequenciais (0, 1) são suficientes. Por que?
- ❖ Precisa verificar se ACK/NAK recebido corrompido.
- ❖ Dobro do número de estados:
 - Estado precisa “lembrar” se pacote “esperado” deve ter $\#seq$ 0 ou 1

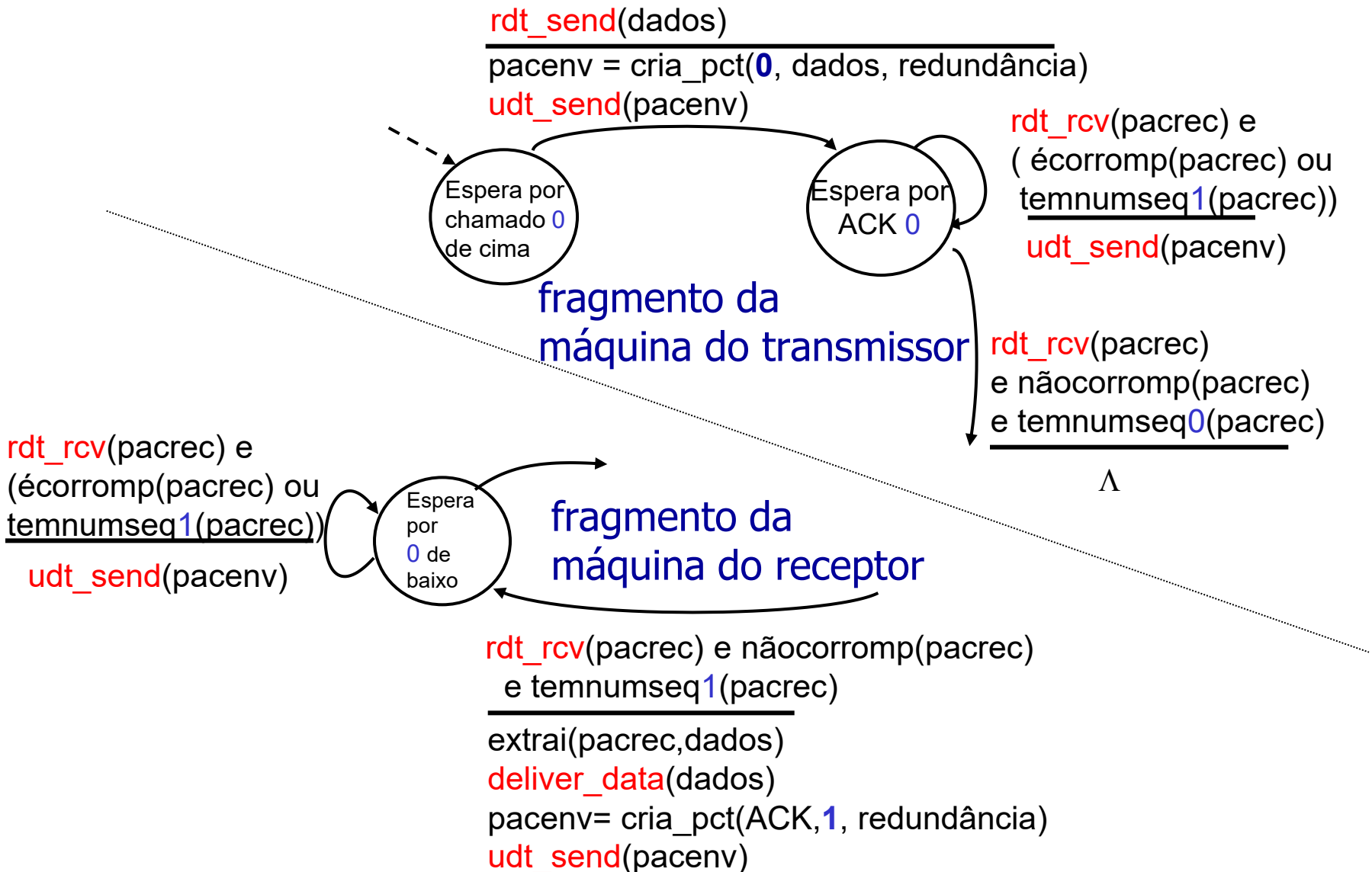
Receptor:

- ❖ Precisa checar se pacote recebido é duplicata:
 - estado indica se 0 ou 1 é $\#seq$ do pacote esperado.
- ❖ Precisa acrescentar redundância no ACK ou NAK.
- ❖ ACK ou NAK não precisa de número sequencial.

rdt2.2: Um protocolo sem NAK

- ❖ Mesma funcionalidade do rdt2.1, usando apenas ACKs.
- ❖ Em vez de NAK, receptor envia ACK para último pacote recebido OK.
 - Receptor precisa incluir *explicitamente* número sequencial do pacote associado ao ACK.
- ❖ ACK duplicado no transmissor resulta na mesma ação do NAK: *retransmite pacote atual*.

rdt2.2: Máquinas para transmissor e receptor



Exercícios interativos

- ❖ Exercício sobre o RDT 2.2 do livro do Kurose