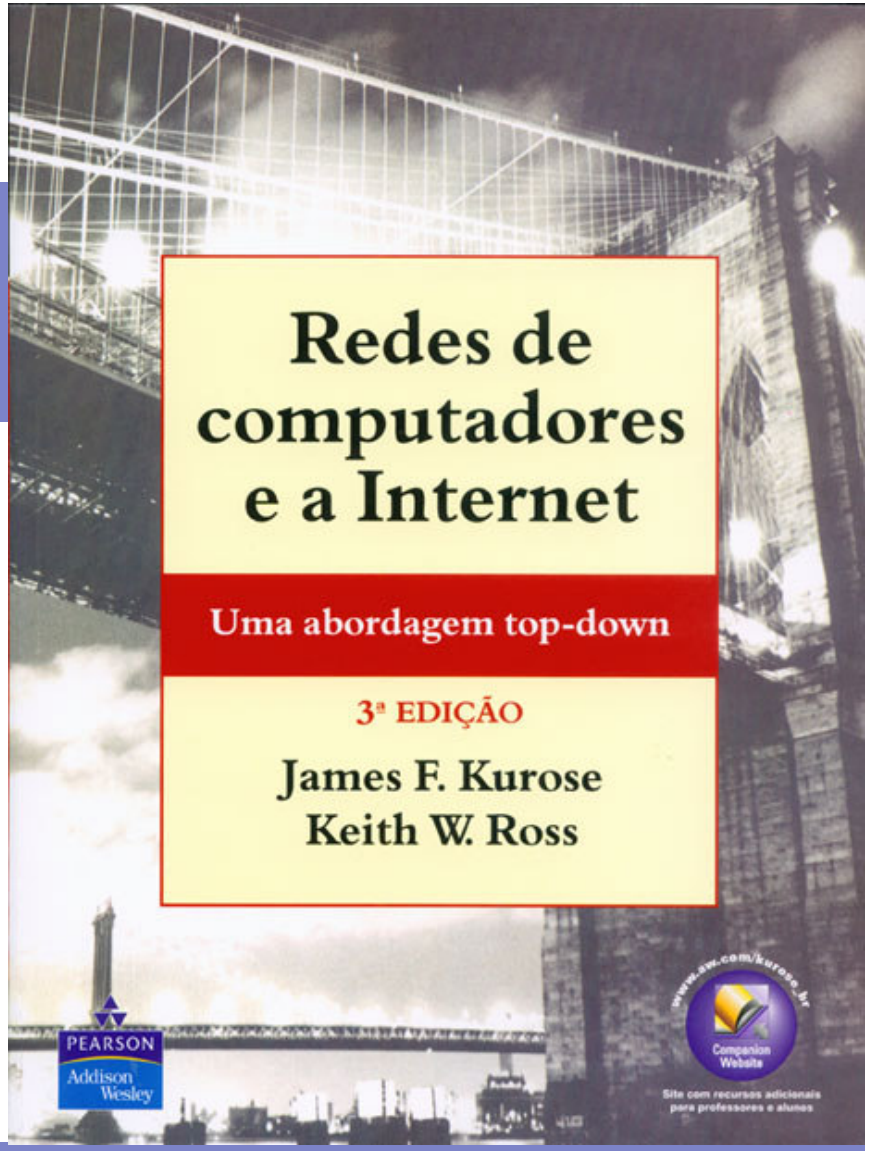
Redes de computadores e a Internet

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 UDP: User Datagram Protocol [RFC 768]

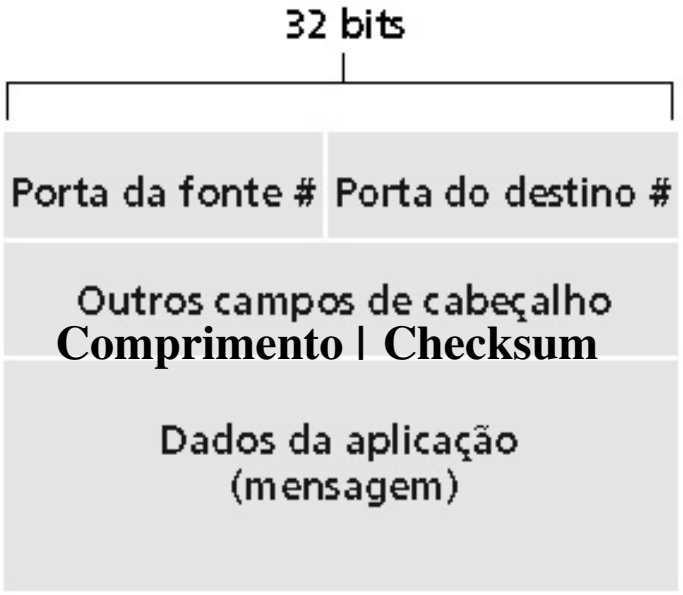
- Protocolo de transporte da Internet “sem gorduras” “sem frescuras”
- Serviço “best effort” , segmentos UDP podem ser:
  - Perdidos
  - Entregues fora de ordem para a aplicação
- Sem conexão:
  - Não há apresentação entre o UDP transmissor e o receptor
  - Cada segmento UDP é tratado de forma independente dos outros

Por que existe um UDP?

- Não há estabelecimento de conexão (que possa redundar em atrasos)
- Simples: não há estado de conexão nem no transmissor, nem no receptor
- Cabeçalho de segmento reduzido
- Não há controle de congestionamento: UDP pode enviar segmentos tão rápido quanto desejado (e possível)

3 Mais sobre UDP

- Muito usado por aplicações de mutimídia contínua (streaming)
  - Tolerantes à perda
  - Sensíveis à taxa
- Outros usos do UDP (por quê?):
  - DNS
  - SNMP
- Transferência confiável sobre UDP: acrescentar confiabilidade na camada de aplicação
  - Recuperação de erro específica de cada aplicação
- Dados da aplicação;
  - 8 – 64Kb



3 UDP checksum

Objetivo: detectar “erros” (ex.: bits trocados) no segmento transmitido

Transmissor:

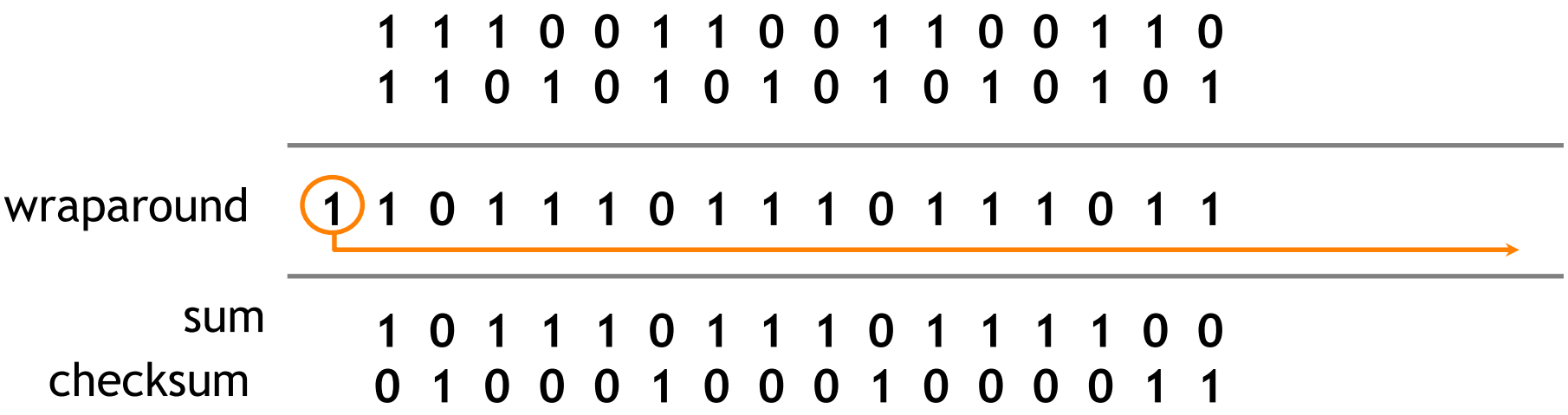
- Trata o conteúdo do segmento como seqüência de inteiros de 16 bits
- Checksum: soma (complemento de 1 da soma) do conteúdo do segmento
- Transmissor coloca o valor do checksum no campo de checksum do UDP

Receptor:

- Computa o checksum do segmento recebido
- Verifica se o checksum calculado é igual ao valor do campo checksum:
  - NÃO - erro detectado
  - SIM - não há erros. **Mas, talvez haja erros apesar disso**? Mas depois…

3 Exemplo: Internet checksum

- Note que:
  - Ao se adicionar números, um *vai um* do bit mais significativo deve ser acrescentado ao resultado.
- Exemplo: adicione dois inteiros de 16 bits



3 Exemplo: Internet checksum

- Exemplo:

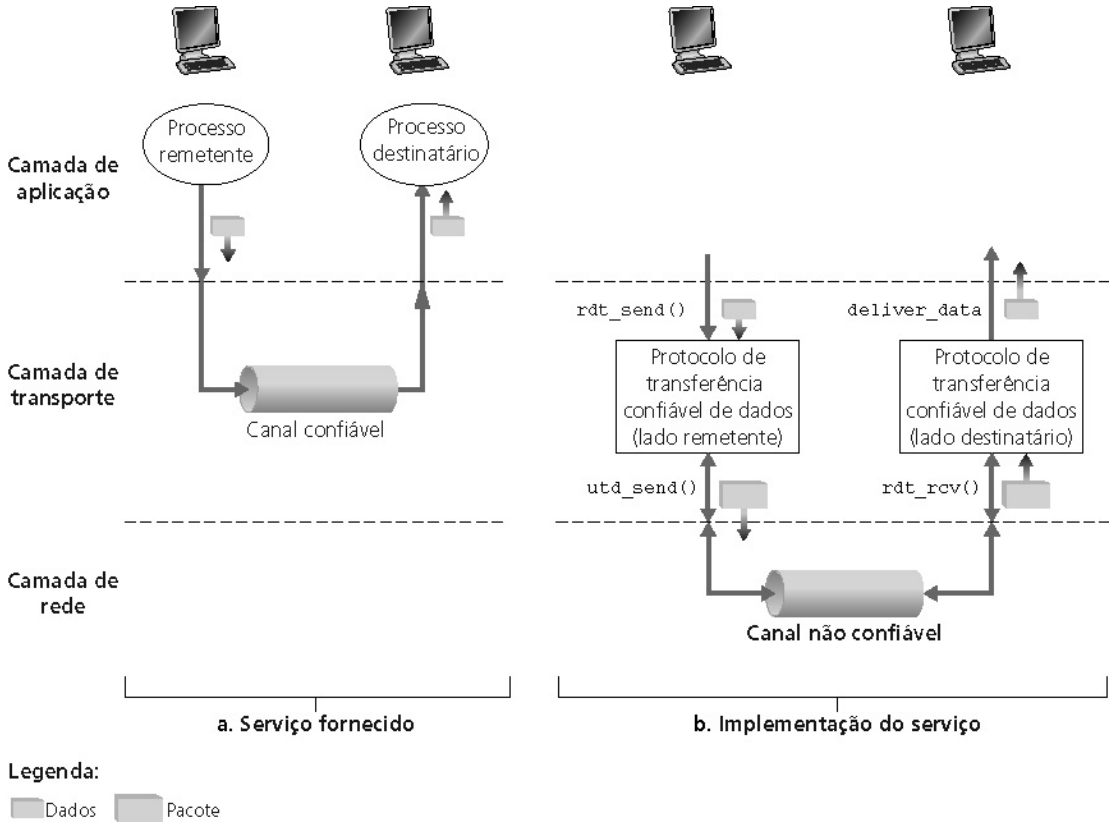
0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0
Checksum?

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

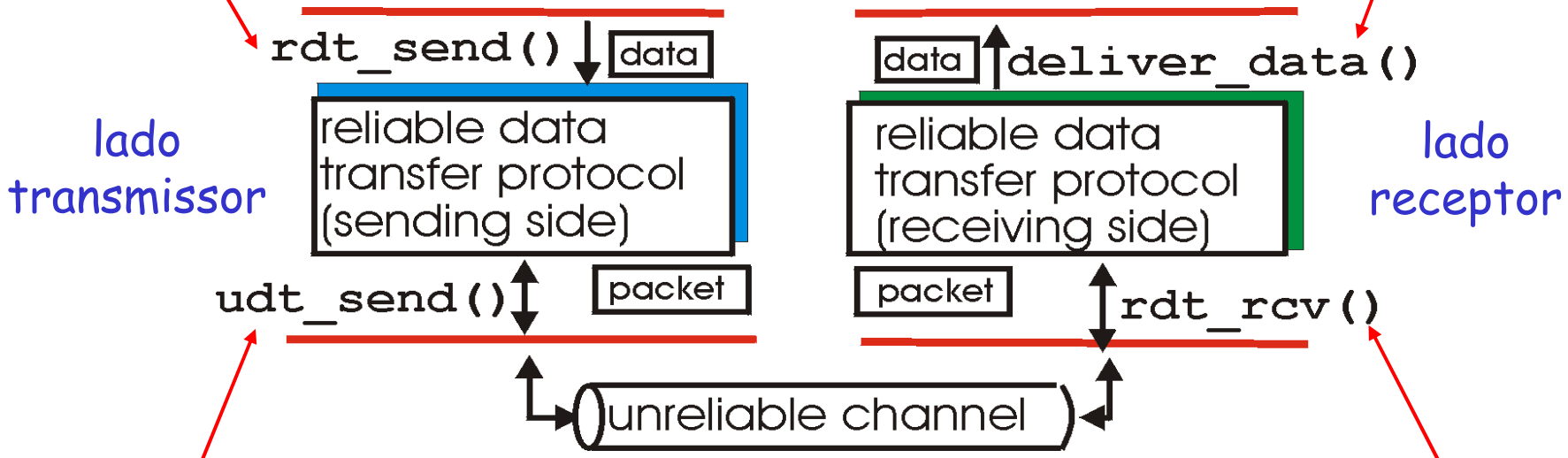
- Importante nas camadas de aplicação, transporte e enlace
- Top 10 na lista dos tópicos mais importantes de redes!
- Características dos canais não confiáveis determinarão a complexidade dos protocolos confiáveis de transferência de dados (rdt)



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

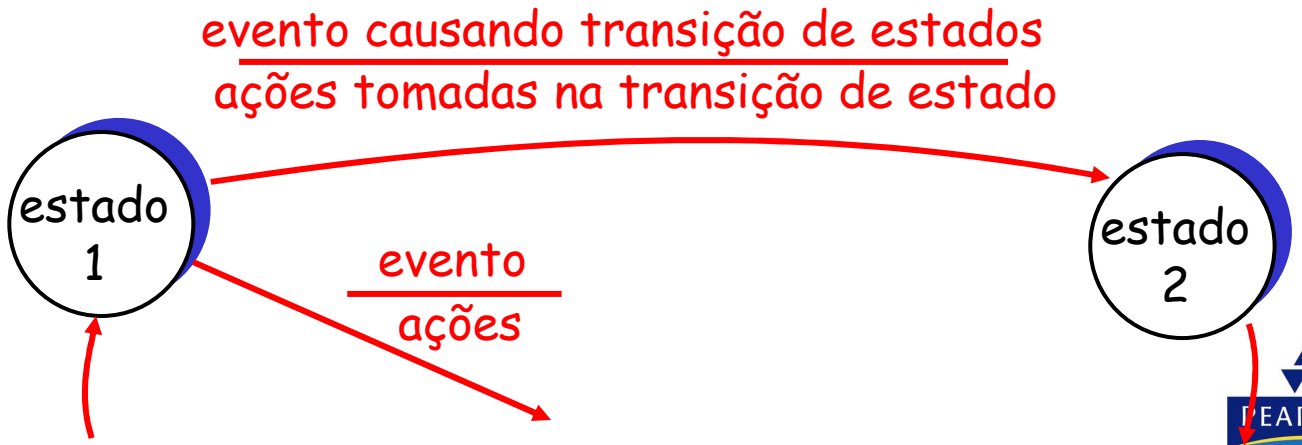
`rdt_rcv()`: chamada quando o pacote chega ao lado receptor do canal

3 Transferência confiável: o ponto de partida

Etapas:

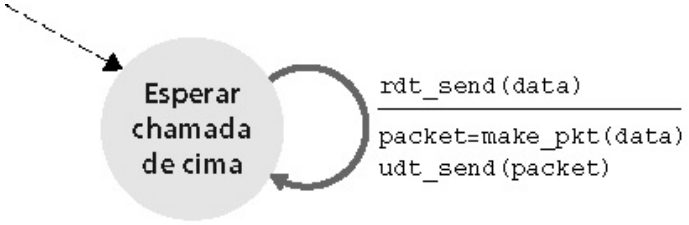
- Desenvolver incrementalmente o transmissor e o receptor de um protocolo confiável de transferência de dados (rdt)
- Considerar apenas transferências de dados unidirecionais
  - Mas informação de controle deve fluir em ambas as direções!
- Usar máquinas de estados finitos (FSM) para especificar o protocolo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

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

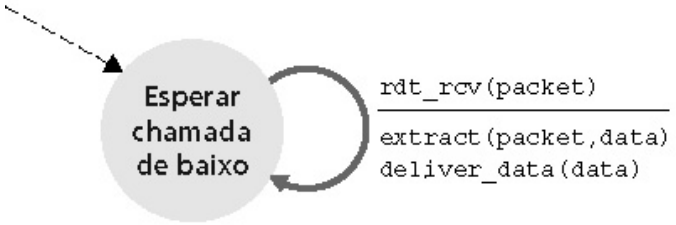


3 rdt1.0: Transfêrencia confiável sobre canais confiáveis

- 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

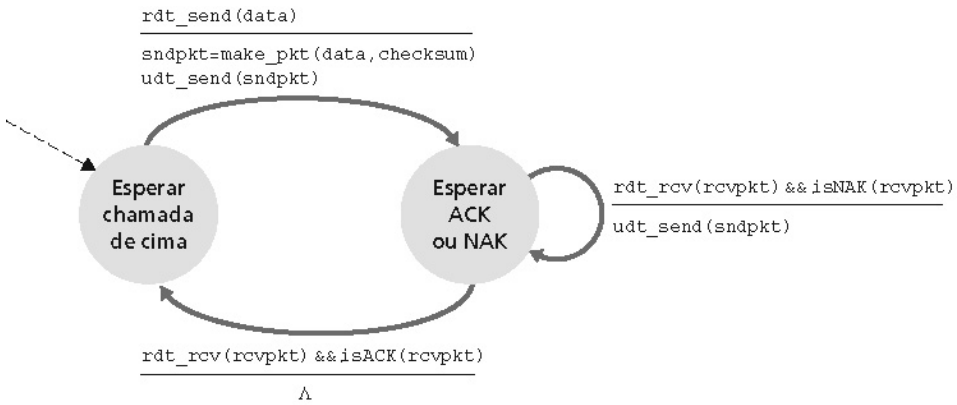


b. rdt1.0: lado destinatário

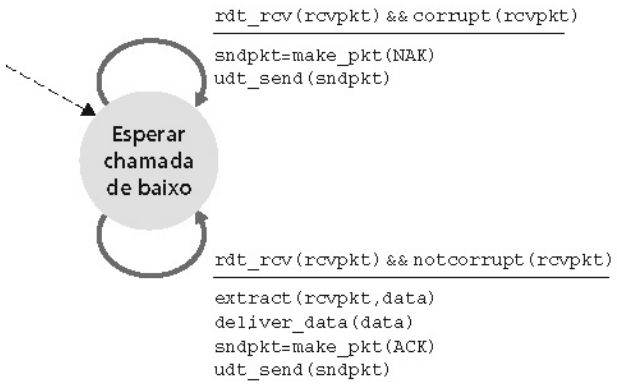
3 rdt2.0: canal com erros de bit

- 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

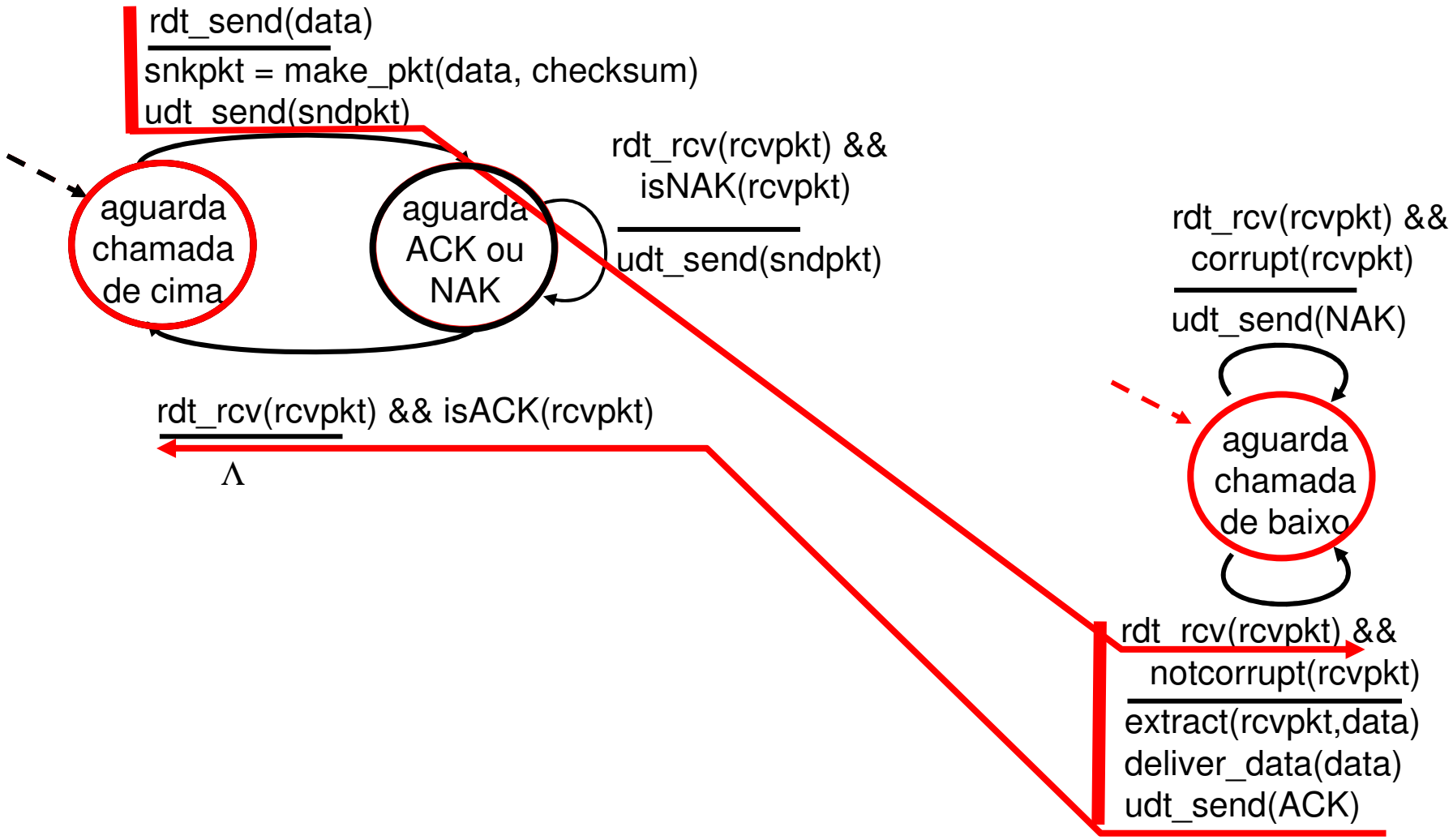


a. rdt2.0: lado remetente

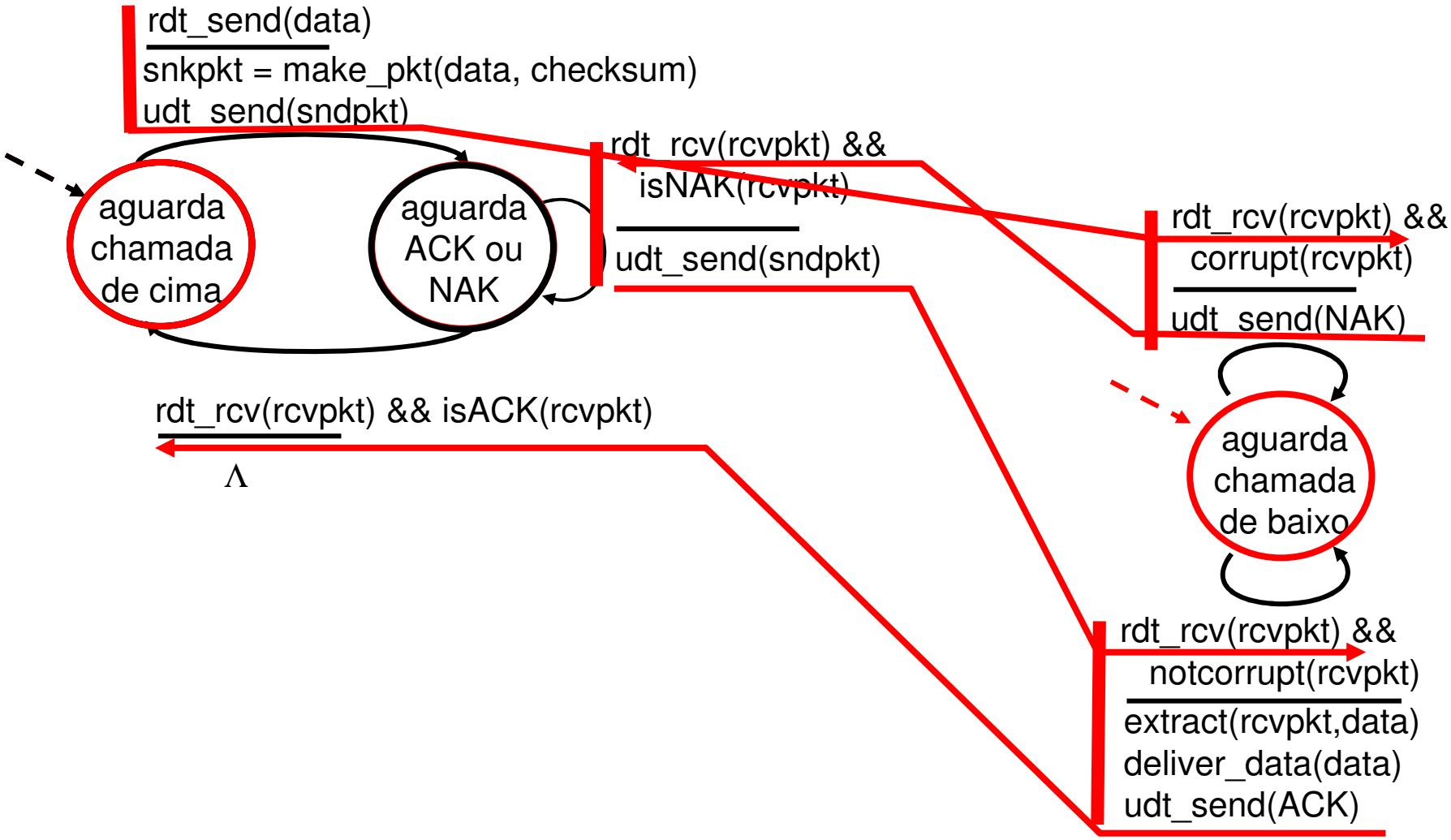


b. rdt2.0: lado destinatário

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!

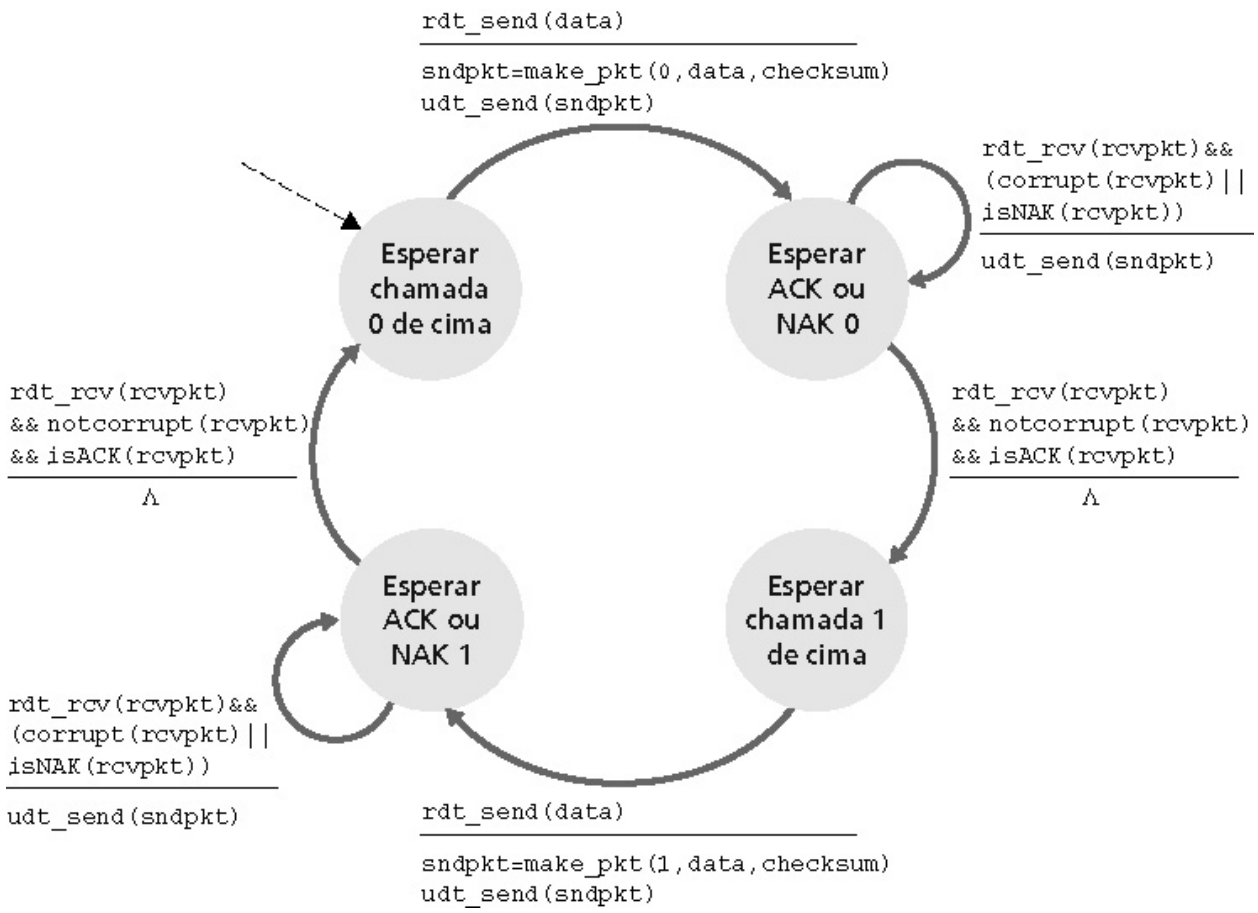
Tratando duplicatas:

- 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

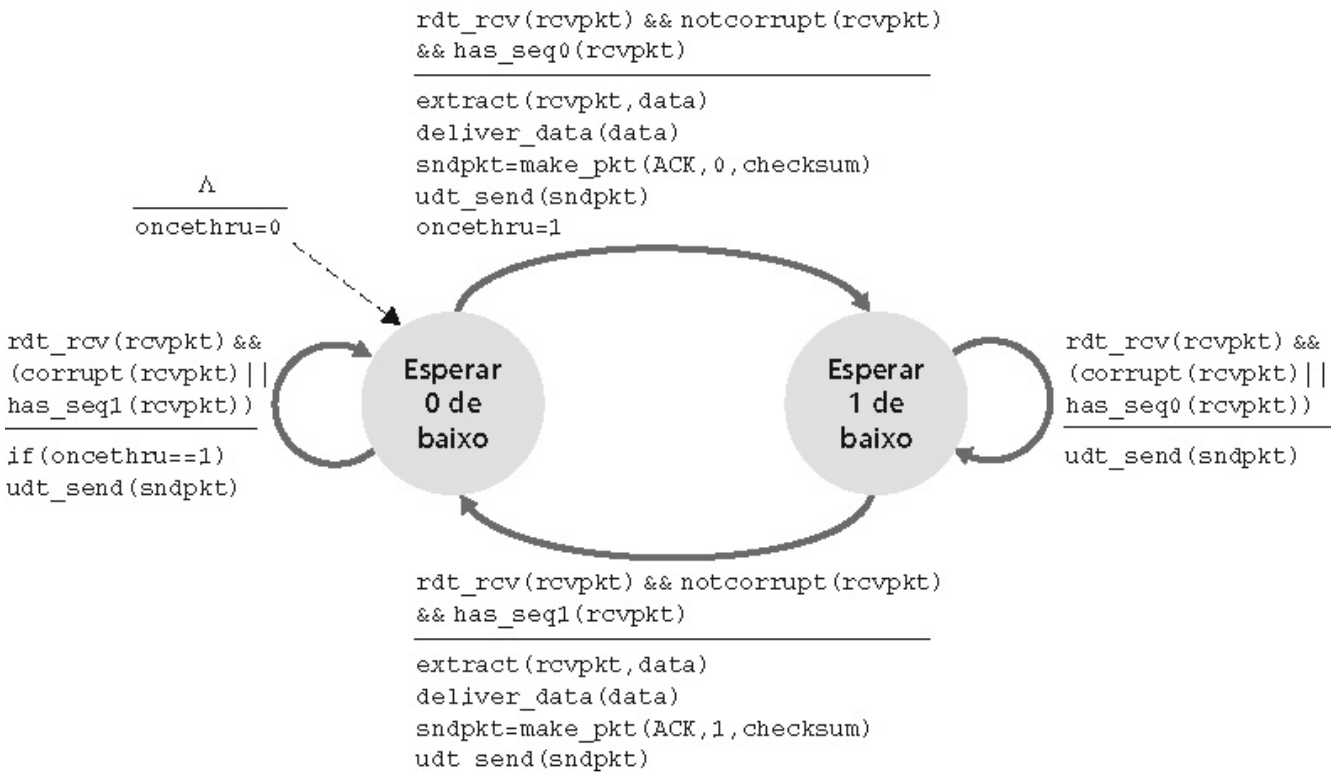
Tipo de protocolo Pare e espere

Transmissor envia um pacotee 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