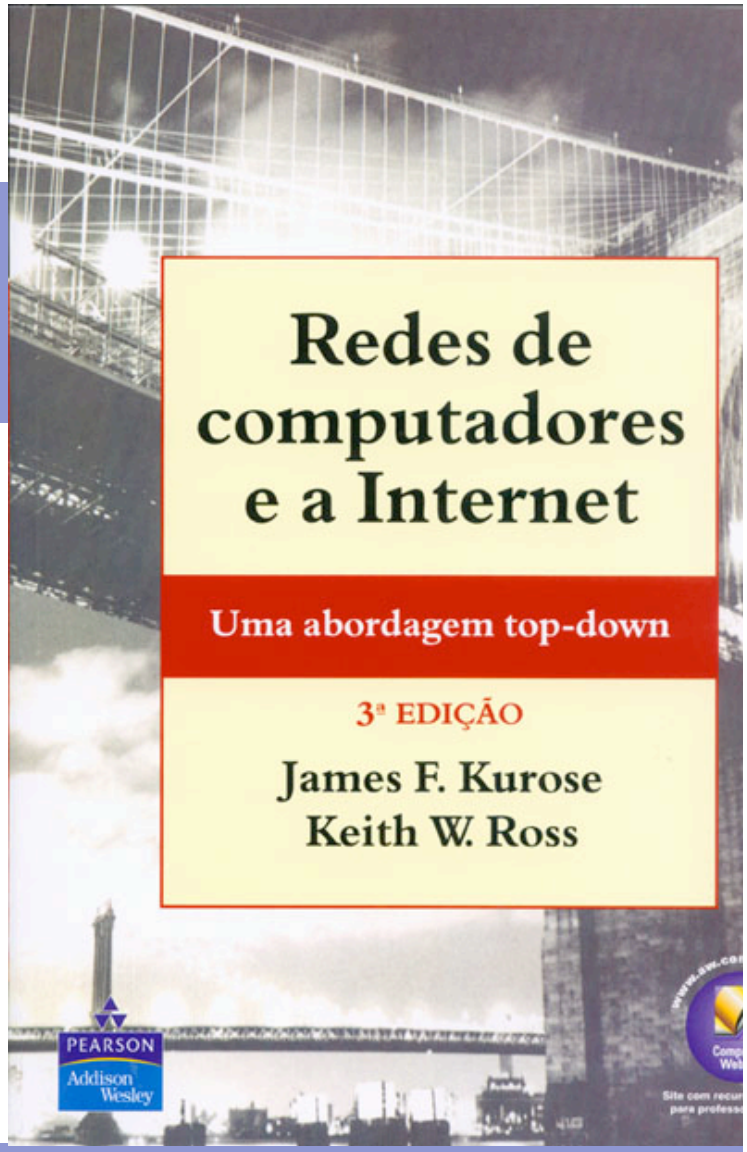
Redes de computadores e a Internet

Capítulo 3

Camada de transporte



3 Camada de transporte

Objetivos do capítulo:

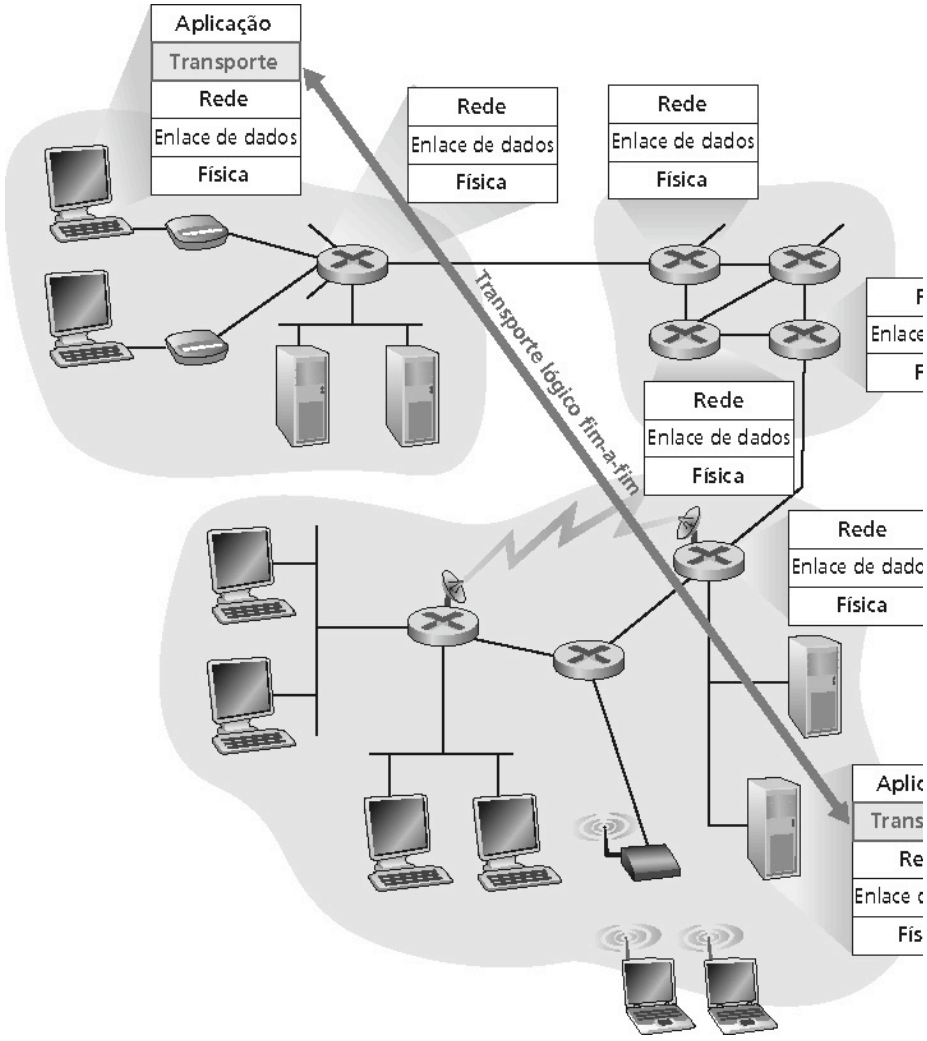
- Entender os princípios por trás dos serviços da camada de transporte:
  - Multiplexação/demultiplexação
  - Transferência de dados confiável
  - Controle de fluxo
  - Controle de congestionamento
- Aprender sobre os protocolos de transporte na Internet:
  - UDP: transporte não orientado à conexão
  - TCP: transporte orientado à conexão
  - Controle de congestionamento do TCP

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 Protocolos e serviços de transporte

- Fornecem comunicação lógica entre processos de aplicação em diferentes hospedeiros
- Os protocolos de transporte são executados nos sistemas finais
  - Lado emissor: quebra as mensagens da aplicação em segmentos e envia para a camada de rede
  - Lado receptor: remonta os segmentos em mensagens e passa para a camada de aplicação
- Há mais de um protocolo de transporte disponível para as aplicações
  - Internet: TCP e UDP



3 Camada de transporte *vs.* camada de rede

- Camada de rede: comunicação lógica entre os hospedeiros
- Camada de transporte: comunicação lógica entre os processos
  - Depende dos serviços da camada de rede

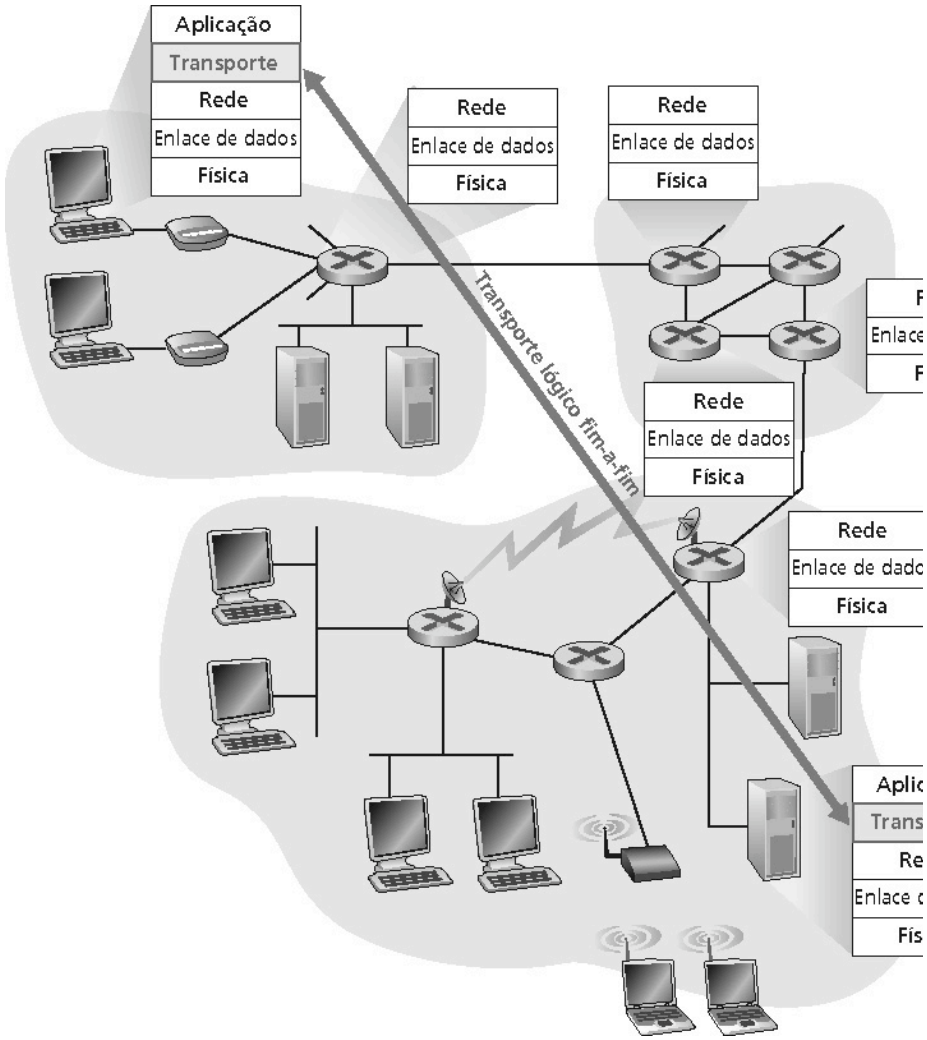
Analogia com uma casa familiar:

12 crianças enviam cartas para 12 crianças

- Processos = crianças
- Mensagens da aplicação = cartas nos envelopes
- Hospedeiros = casas
- Protocolo de transporte = Anna e Bill
- Protocolo da camada de rede = serviço postal

3 Protocolos da camada de transporte da Internet

- Confiável, garante ordem de entrega (TCP)
- Controle de congestionamento
  - Controle de fluxo
  - Orientado à conexão
- Não confiável, sem ordem de entrega: UDP
  - Extensão do “melhor esforço” do IP
- Serviços não disponíveis:
  - Garantia a atrasos
  - Garantia de banda



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

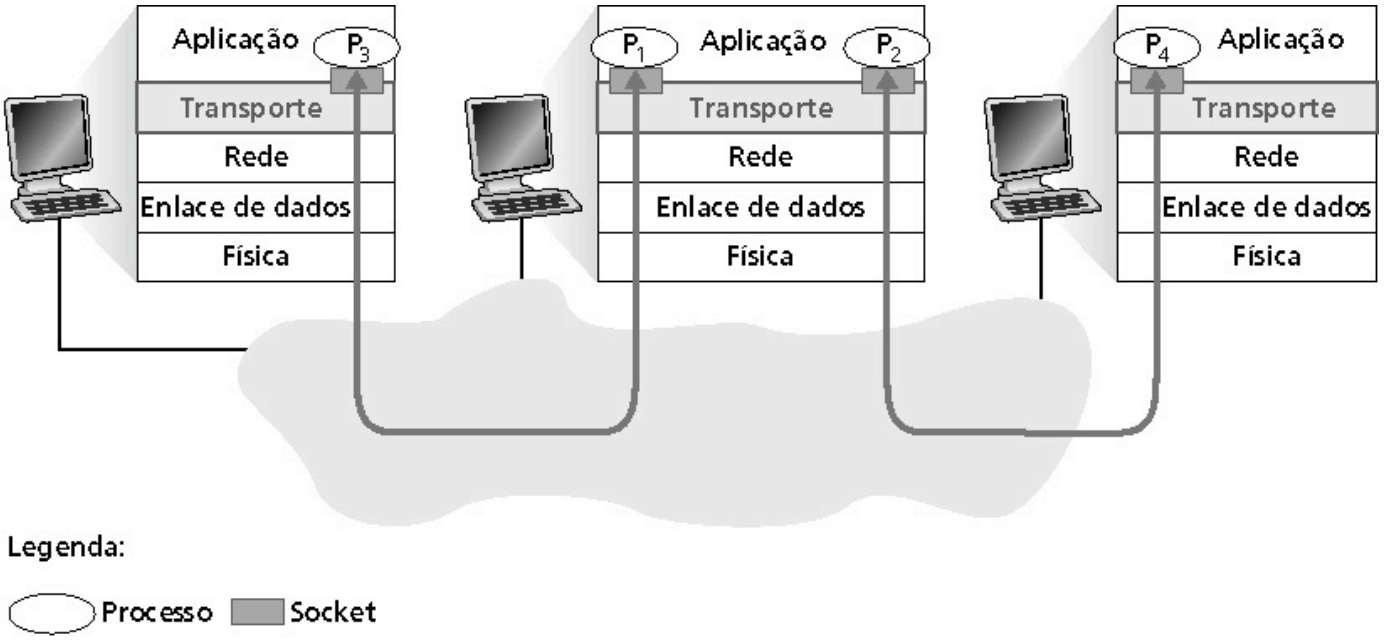
3 Multiplexação/demultiplexação

Demultiplexação no hospedeiro receptor:

entrega os segmentos recebidos ao socket correto

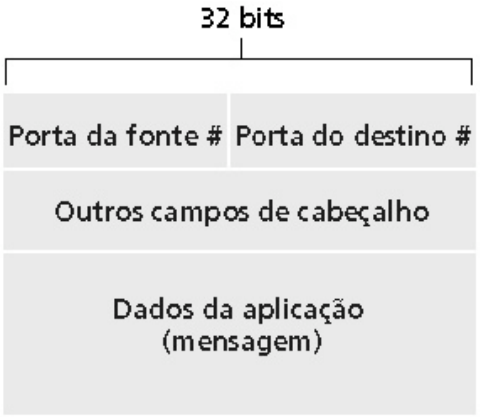
Multiplexação no hospedeiro emiss

coleta dados de múltiplos sockets, envelopa os dados com cabeçalho (usado depois para demultiplexaç



3 Como funciona a demultiplexação

- Computador recebe datagramas IP
  - Cada datagrama possui endereço IP de origem e IP de destino
  - Cada datagrama carrega 1 segmento da camada de transporte
  - Cada segmento possui números de porta de origem e destino (lembre-se: números de porta bem conhecidos para aplicações específicas)
- O hospedeiro usa endereços IP e números de porta para direcionar o segmento ao socket apropriado



3 Demultiplexação não orientada à conexão

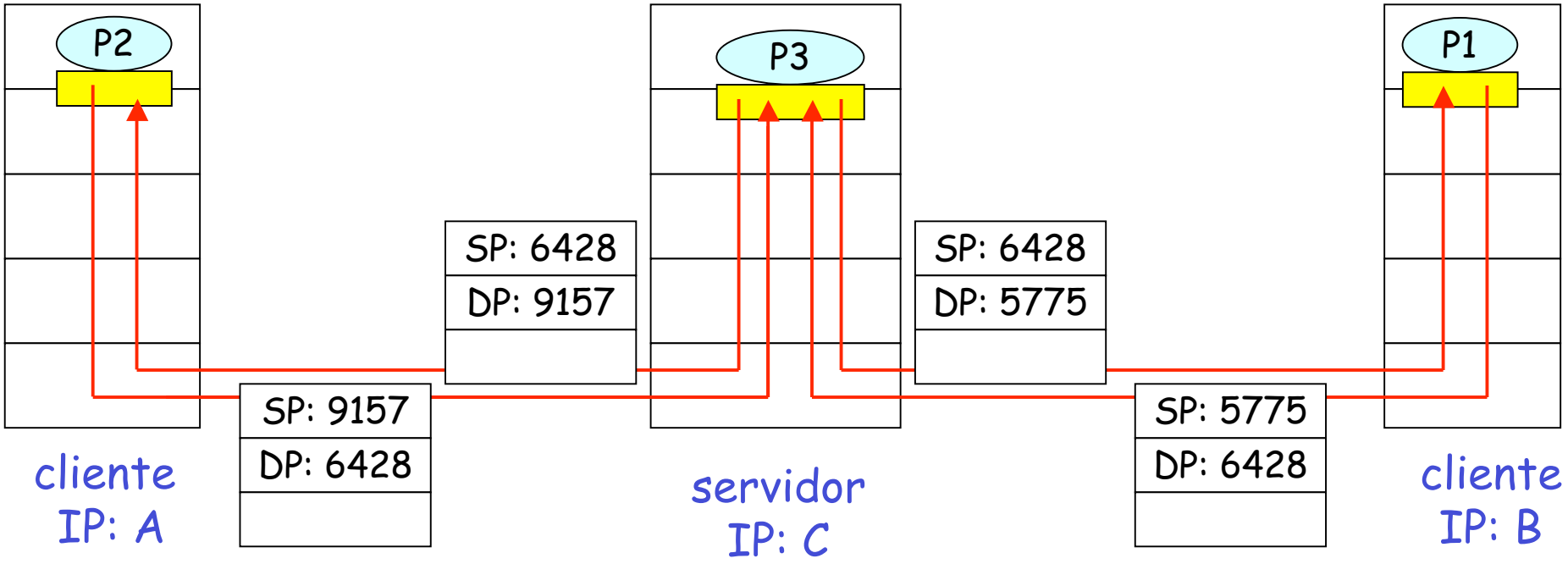
- Cria sockets com números de porta:

```
DatagramSocket mySocket1 = new DatagramSocket(99111);
DatagramSocket mySocket2 = new DatagramSocket(99222);
```

- Socket UDP identificado por dois valores:
  (endereço IP de destino, número da porta de destino)
- Quando o hospedeiro recebe o segmento UDP:
  - Verifica o número da porta de destino no segmento
  - Direciona o segmento UDP para o socket com este número de porta
- Datagramas com IP de origem diferentes e/ou portas de origem diferentes são direcionados para o mesmo socket

3 Demultiplexação não orientada à conexão

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

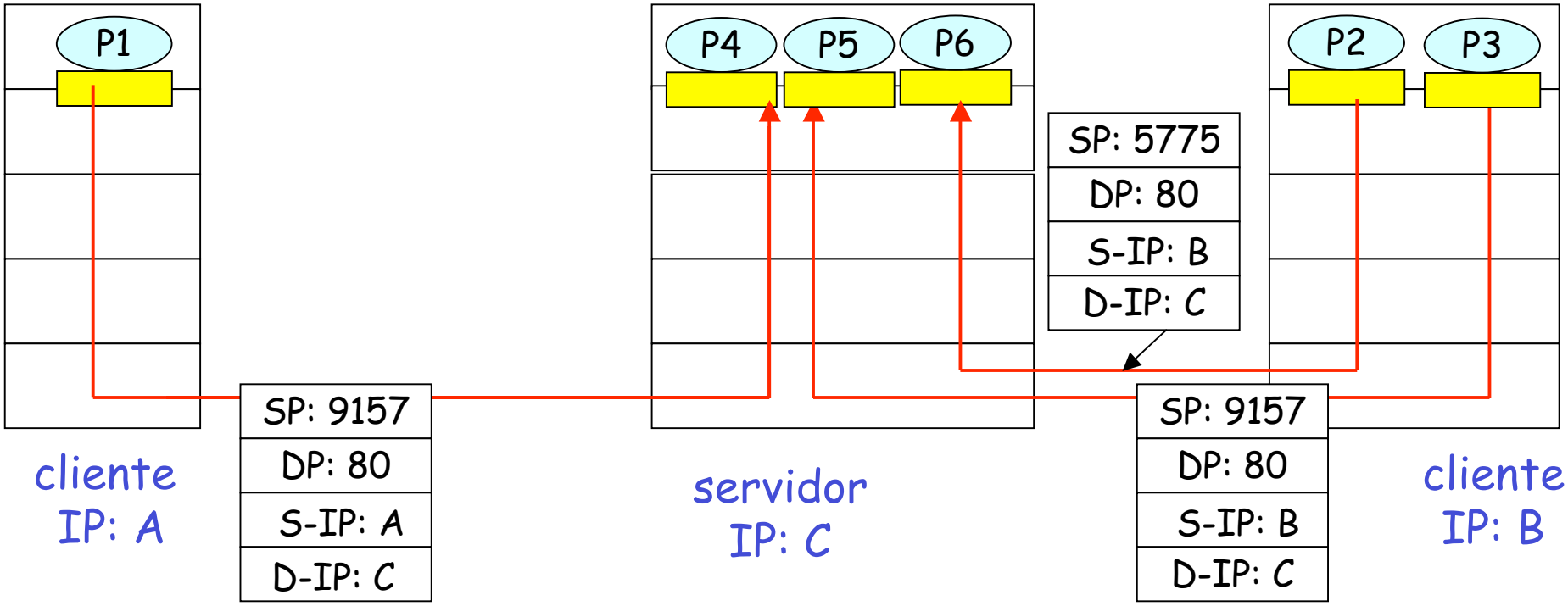


SP fornece o "endereço retorno" 0

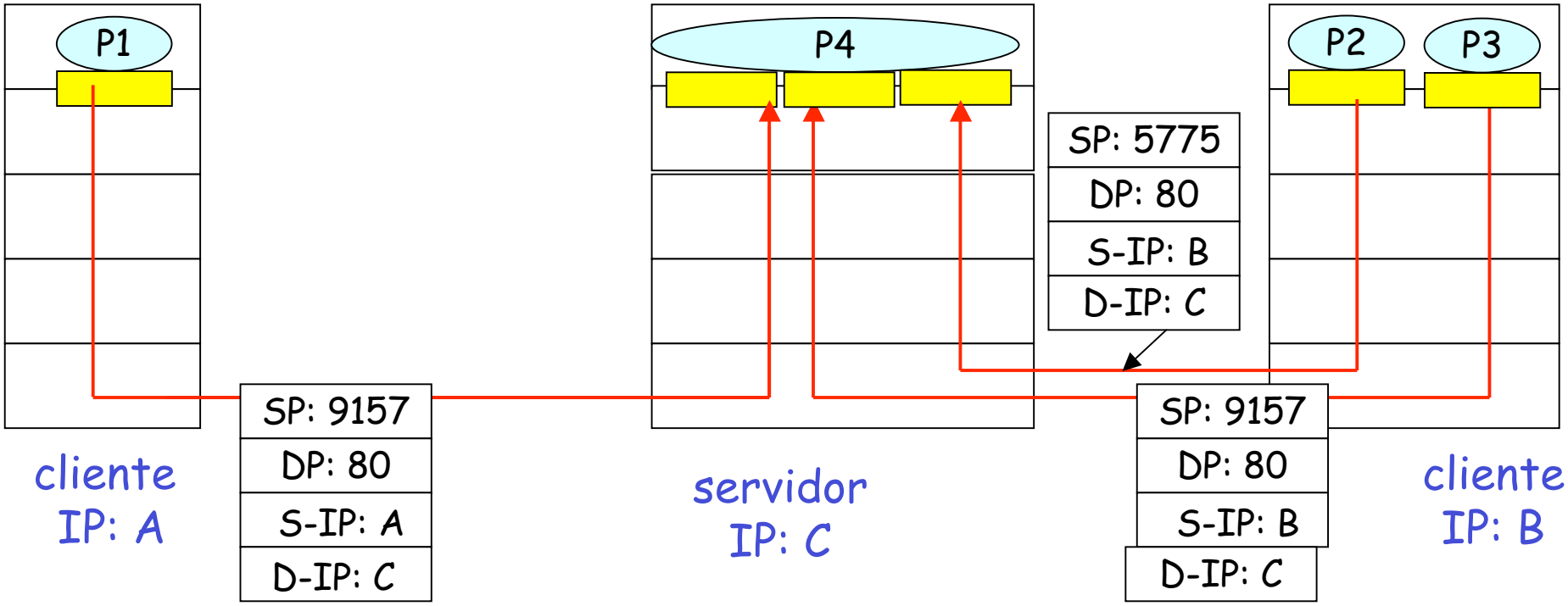
3 Demux orientada à conexão

- Socket TCP identificado por 4 valores:
  - Endereço IP de origem
  - End. porta de origem
  - Endereço IP de destino
  - End. porta de destino
- Hospedeiro receptor usa os quatro valores para direcionar o segmento ao socket apropriado
- Hospedeiro servidor pode suportar vários sockets TCP simultâneos:
  - Cada socket é identificado pelos seus próprios 4 valores
  - Servidores Web possuem sockets diferentes para cada cliente conectado
  - HTTP não persistente terá um socket diferente para cada requisição

3 Demux orientada à conexão



3 Demux orientada à conexão servidor Web “threaded”



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]

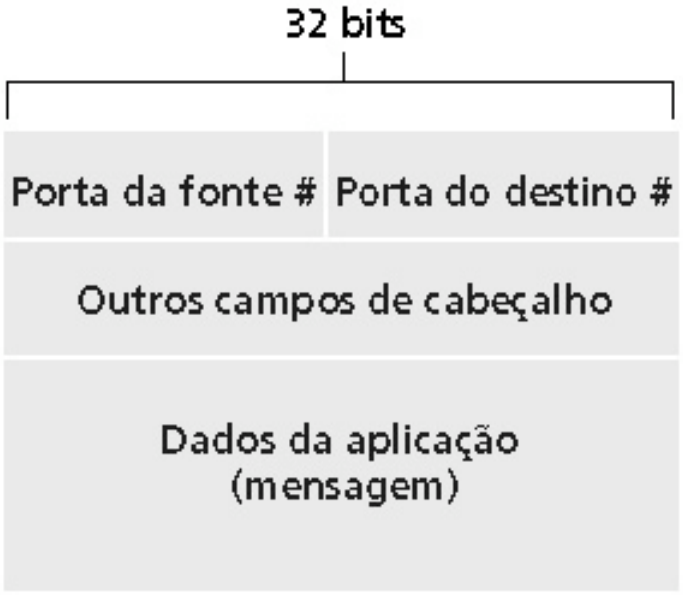
- Protocolo de transporte da Internet “sem gorduras”, “sem frescuras”
- Serviço “best effort”, segmentos UDP podem ser:
  - Perdidos
  - Entregues fora de ordem para a aplicação
- Sem conexão:
  - Não há apresentação entre o UDP transmissor e o receptor
  - Cada segmento UDP é tratado de forma independente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



3 UDP checksum

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

Transmissor:

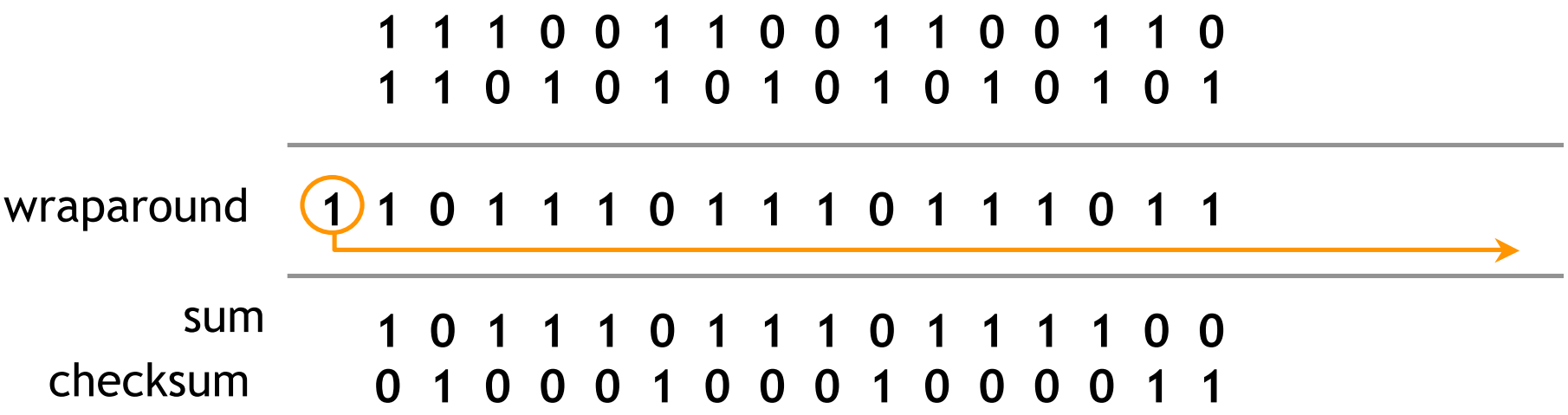
- Trata o conteúdo do segmento como seqüência de inteiros de 16 bits
- Checksum: soma (complemento de 1 da soma) do conteúdo do segmento
- Transmissor coloca o valor do checksum no campo de checksum do UDP

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

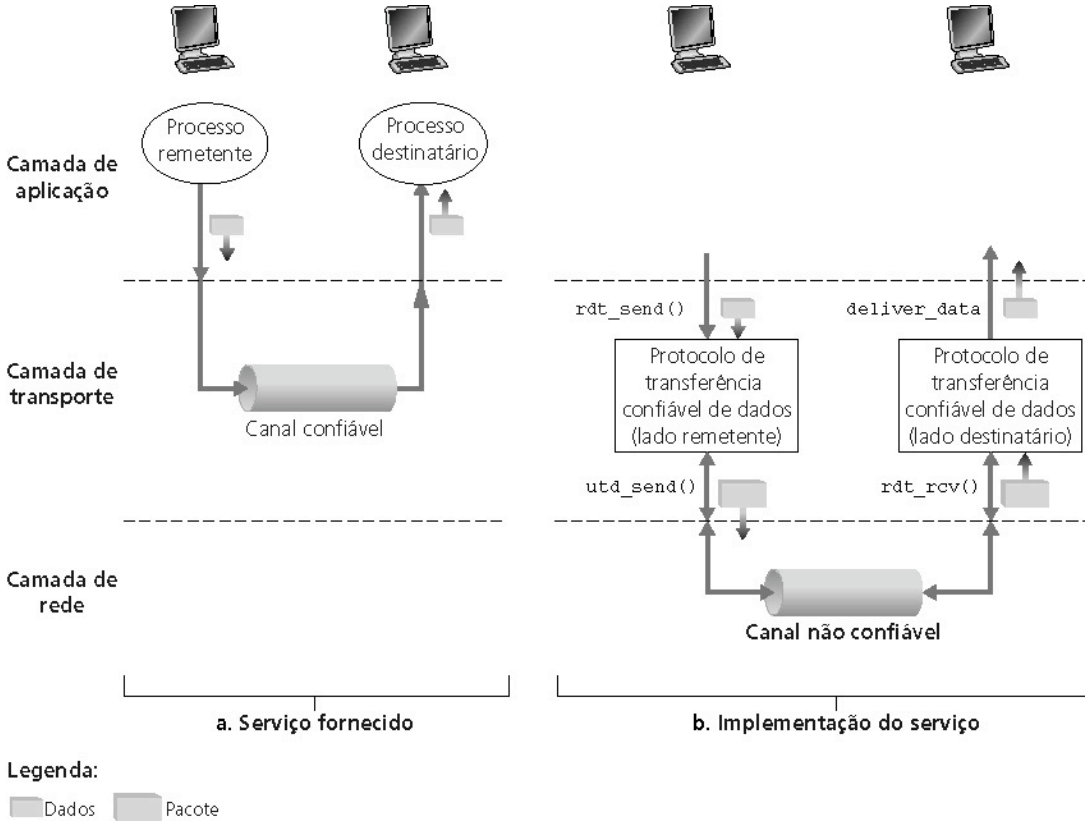


3 Camada de transporte

- 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

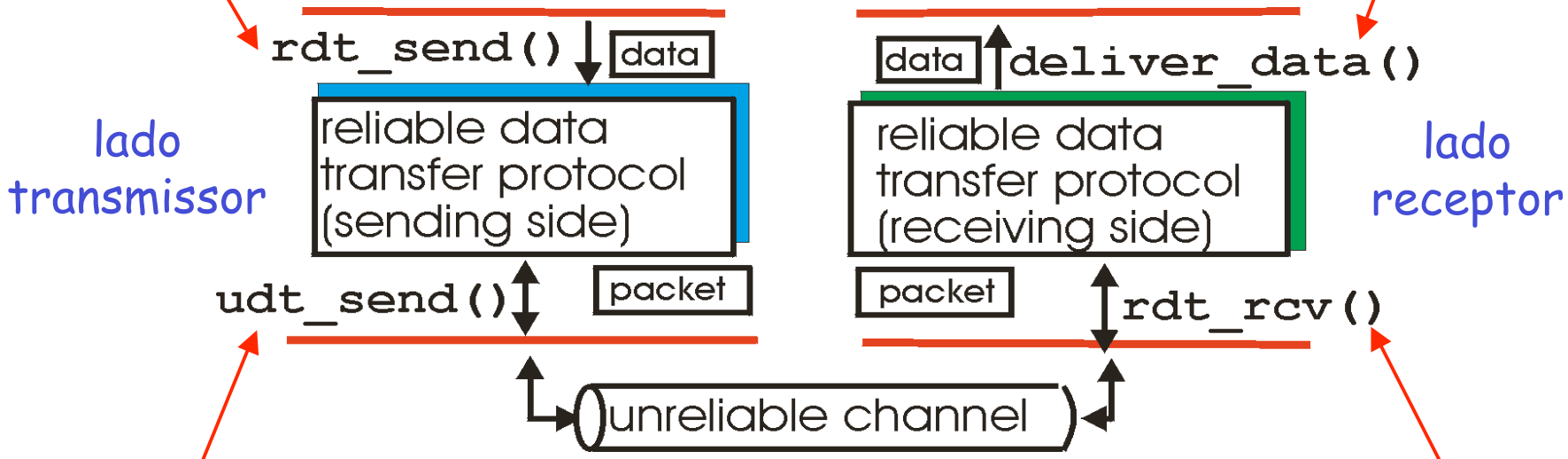
- Importante nas camadas de aplicação, transporte e enlace
- Top 10 na lista dos tópicos mais importantes de redes!
- Características dos canais não confiáveis determinarão a complexidade dos protocolos confiáveis de transferência de dados (rdt)



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

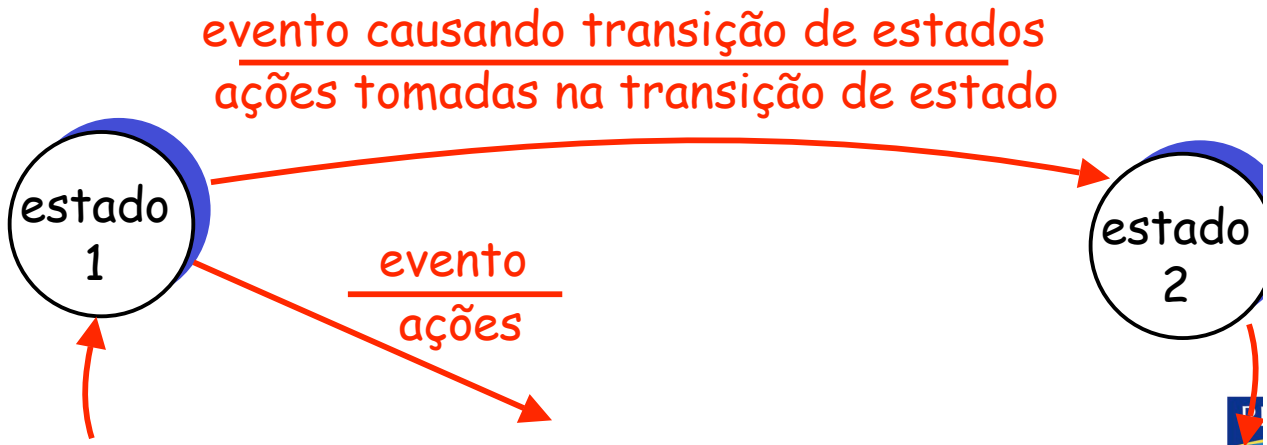
`rdt_rcv()`: chamada quando o pacote chega ao lado receptor do canal

3 Transferência confiável: o ponto de partida

Etapas:

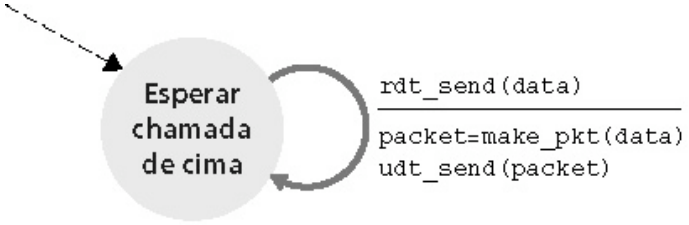
- Desenvolver incrementalmente o transmissor e o receptor de um protocolo confiável de transferência de dados (rdt)
- Considerar apenas transferências de dados unidirecionais
  - Mas informação de controle deve fluir em ambas as direções!
- Usar máquinas de estados finitos (FSM) para especificar o protocolo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

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

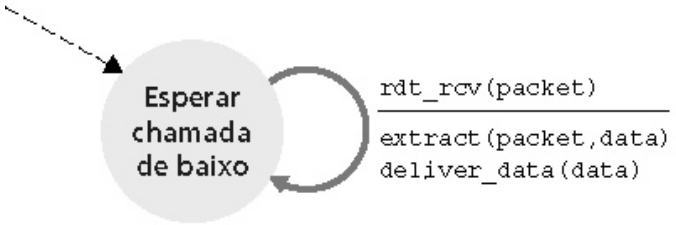


3 rdt1.0: Transferência confiável sobre canais confiáveis

- 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

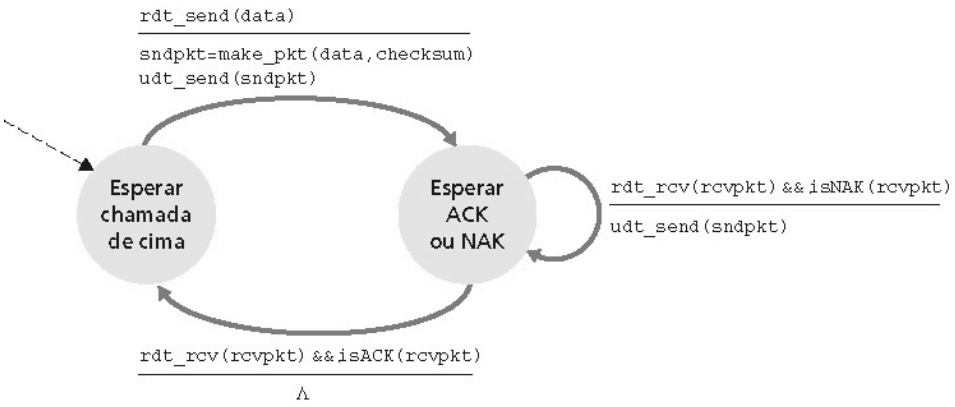


b. rdt1.0: lado destinatário

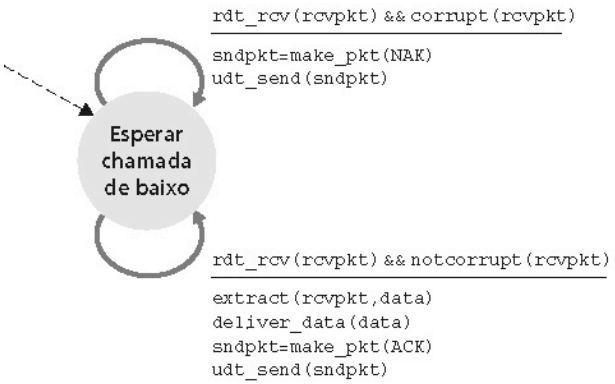
3 rdt2.0: canal com erros de bit

- 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

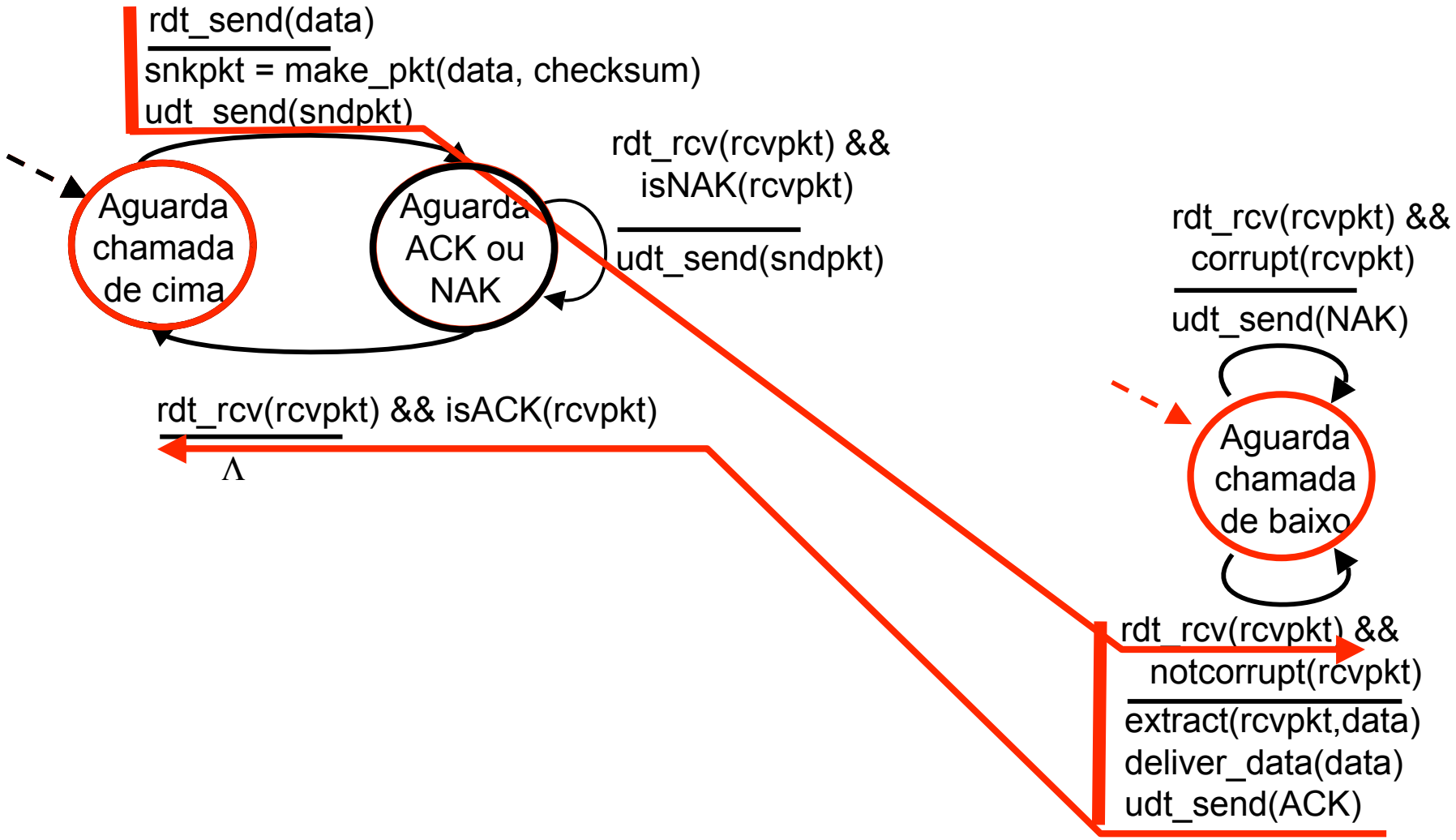


a. rdt2.0: lado remetente

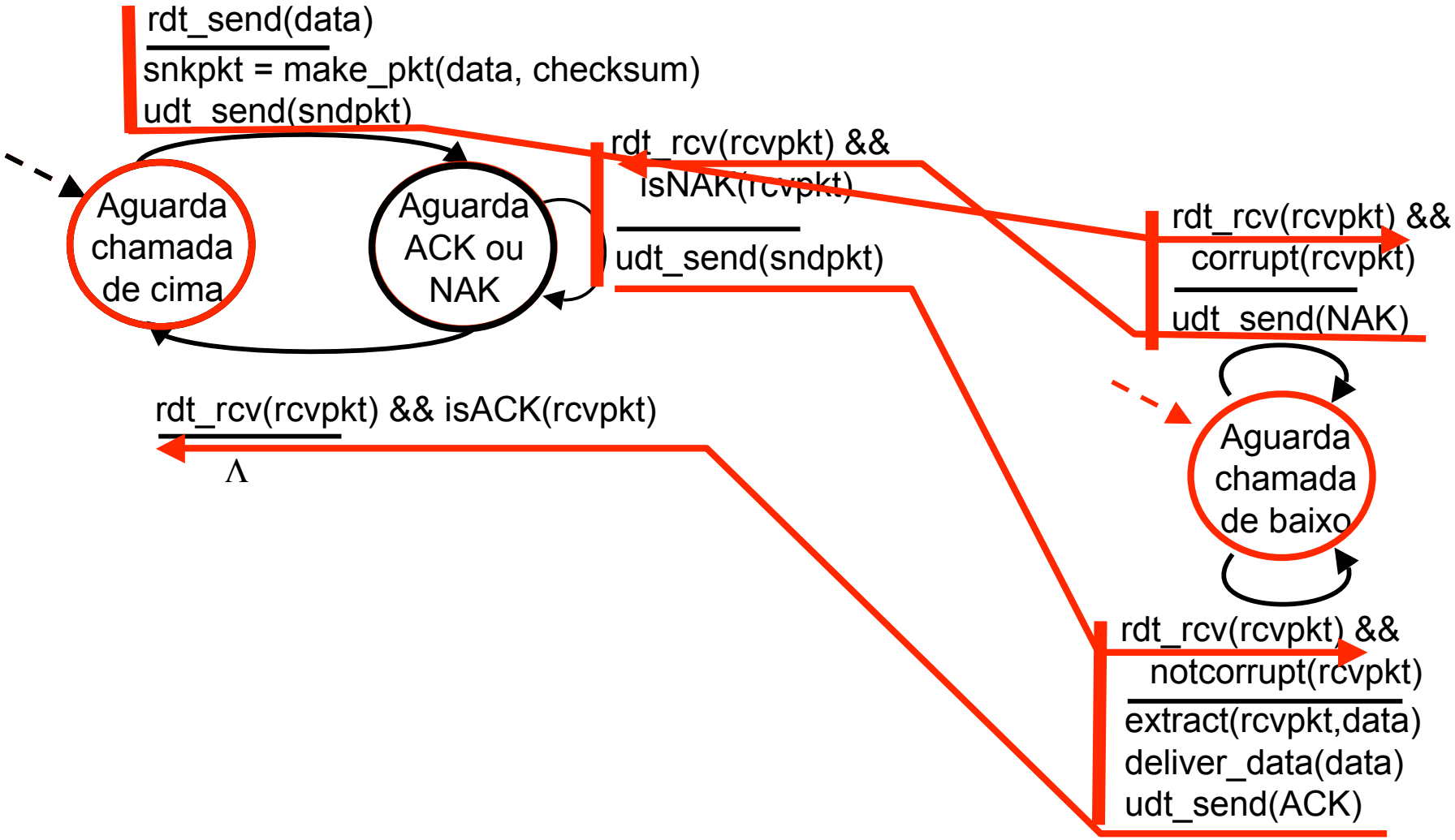


b. rdt2.0: lado destinatário

3 rdt2.0 operação com ausência de erros



3 rdt2.0: cenário de erro



3 rdt2.0 tem um problema fatal!

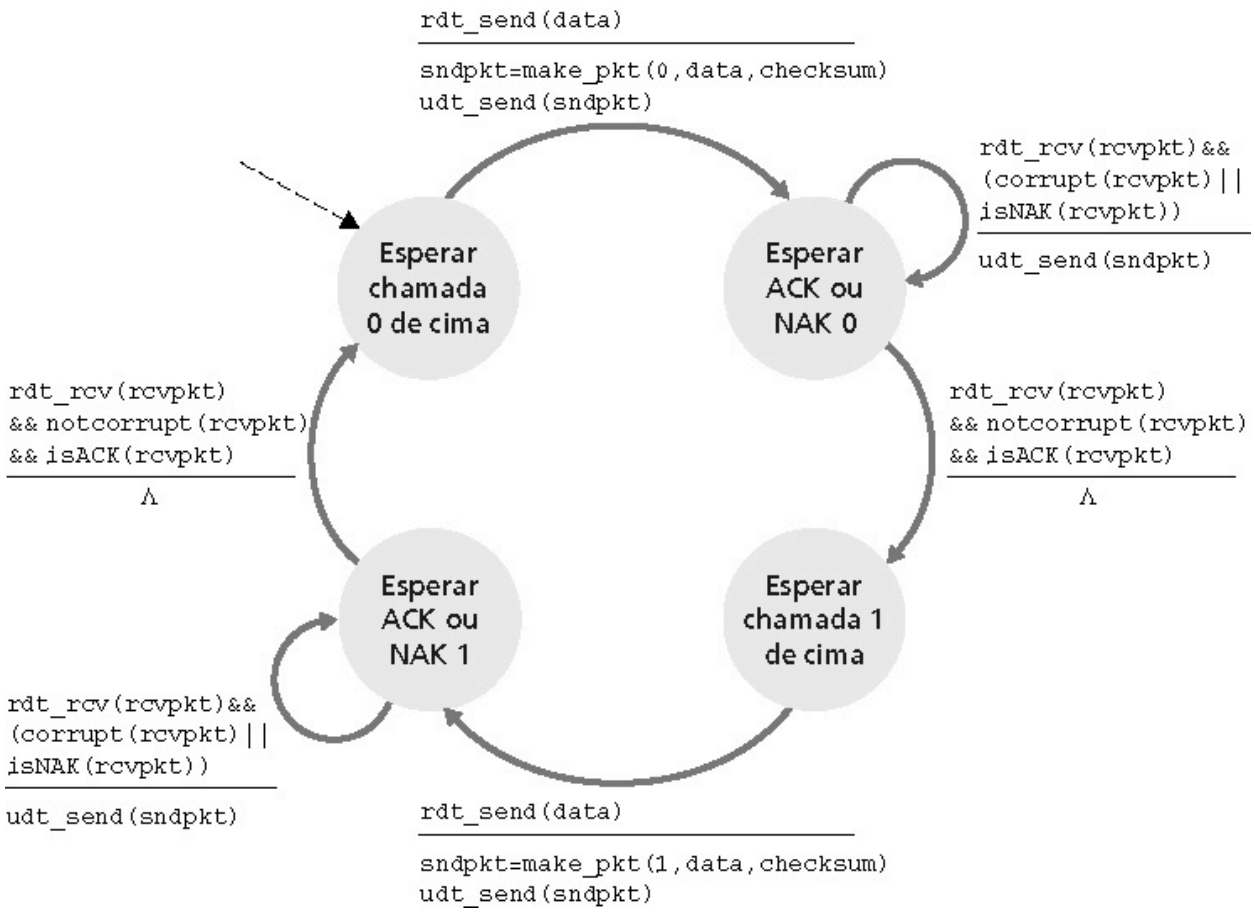
O que acontece se o ACK/NAK é corrompido?
- Transmissor não sabe o que aconteceu no receptor!
- Não pode apenas retransmitir: possível duplicata

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

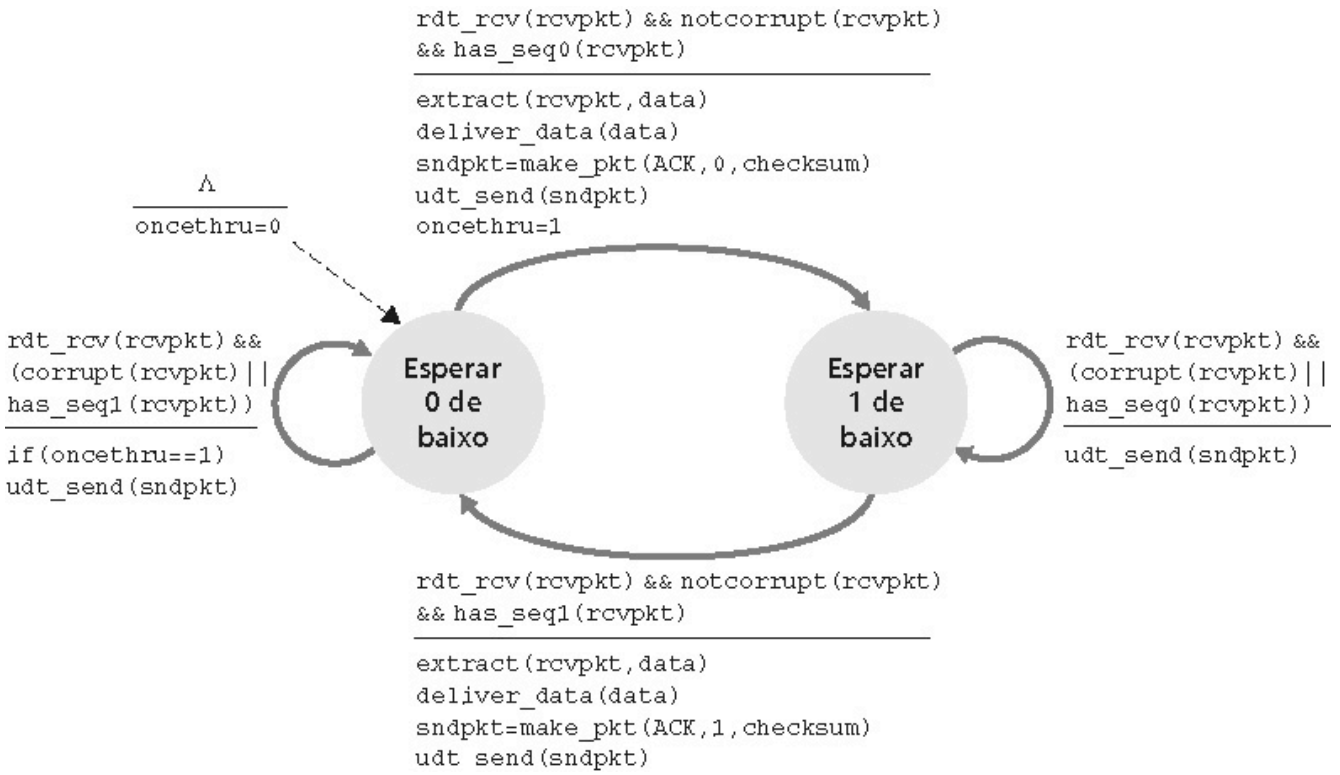
Pare e espere

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

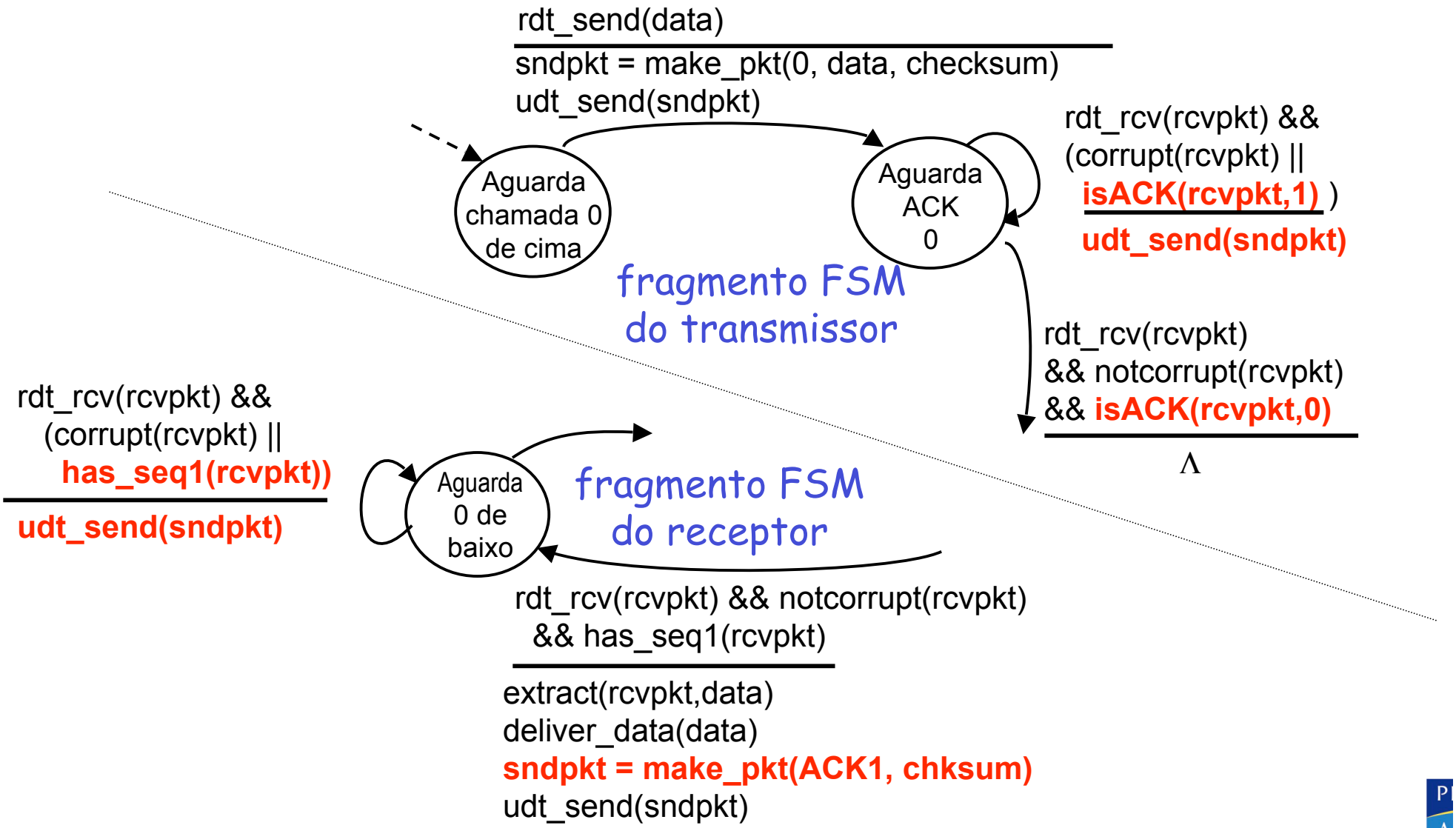
Receptor:

- 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

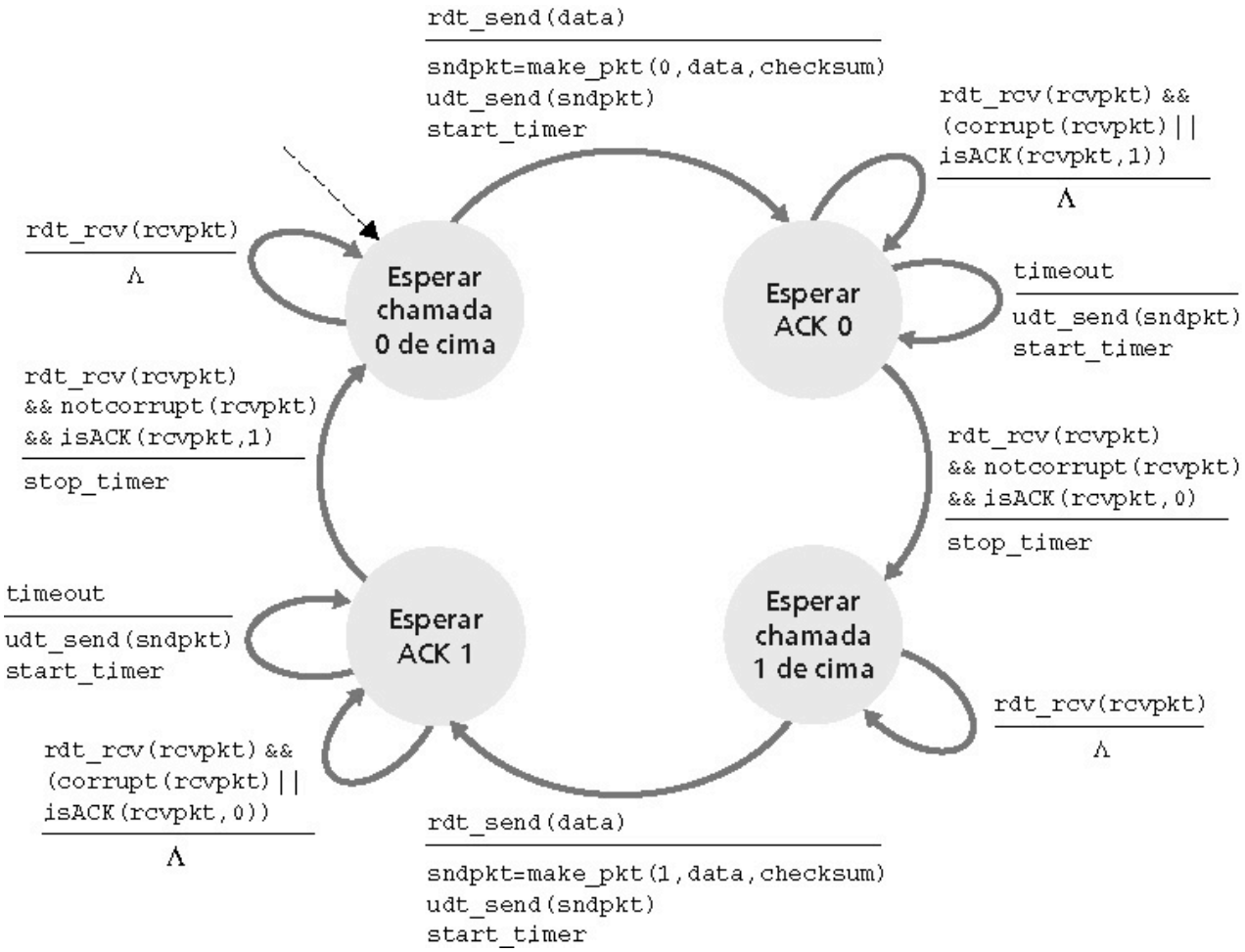
Nova hipótese: canal de transmissão pode também perder pacotes (devido aos ACKs)

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

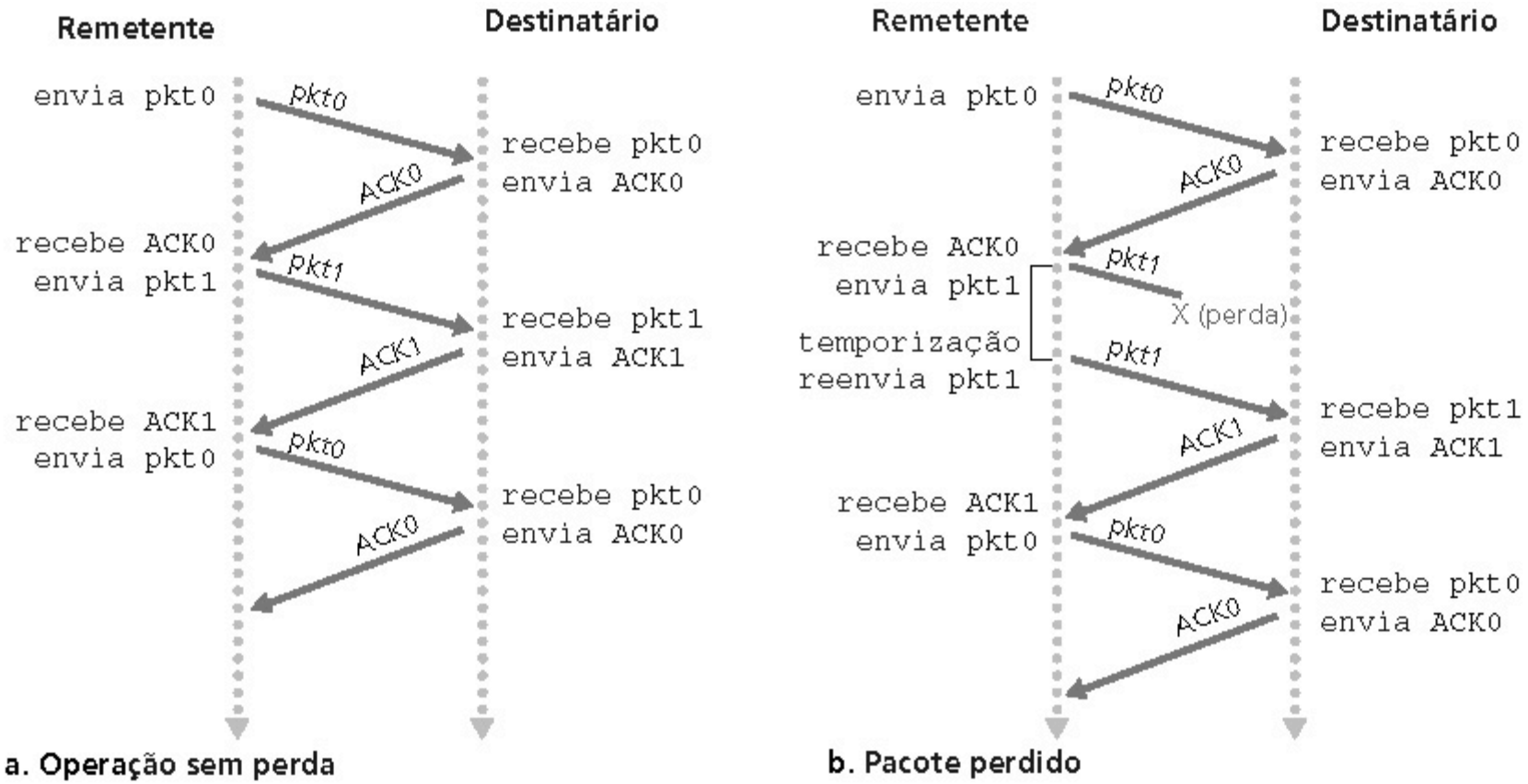
Abordagem: transmissor espera um tempo "razoável" pelo ACK

- 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

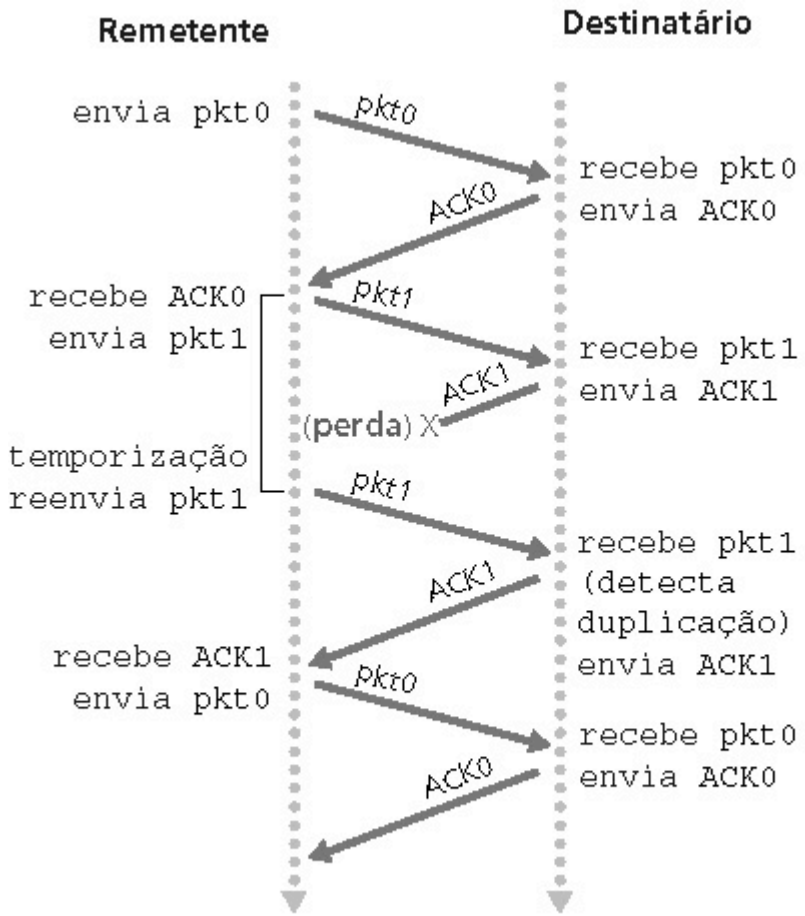
3 Transmissor rdt3.0



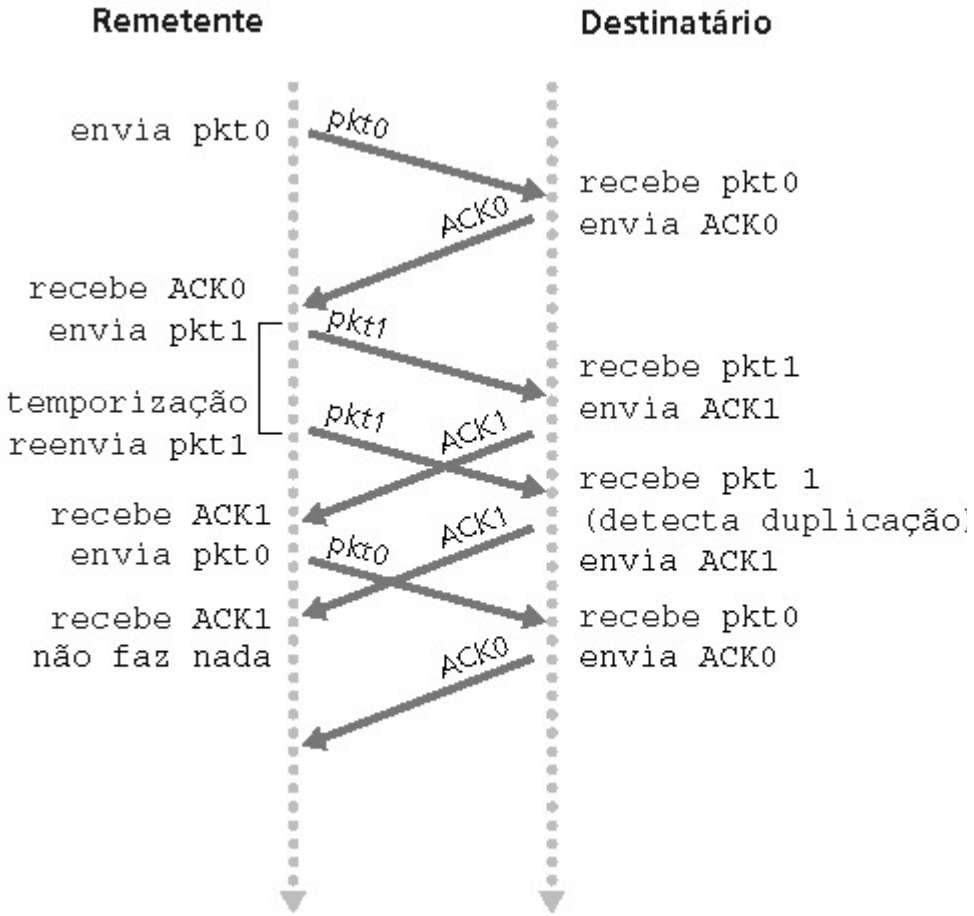
3 rdt3.0 em ação



3 rdt3.0 em ação



c. ACK perdido



d. Interrupção prematura

3 Desempenho do rdt3.0

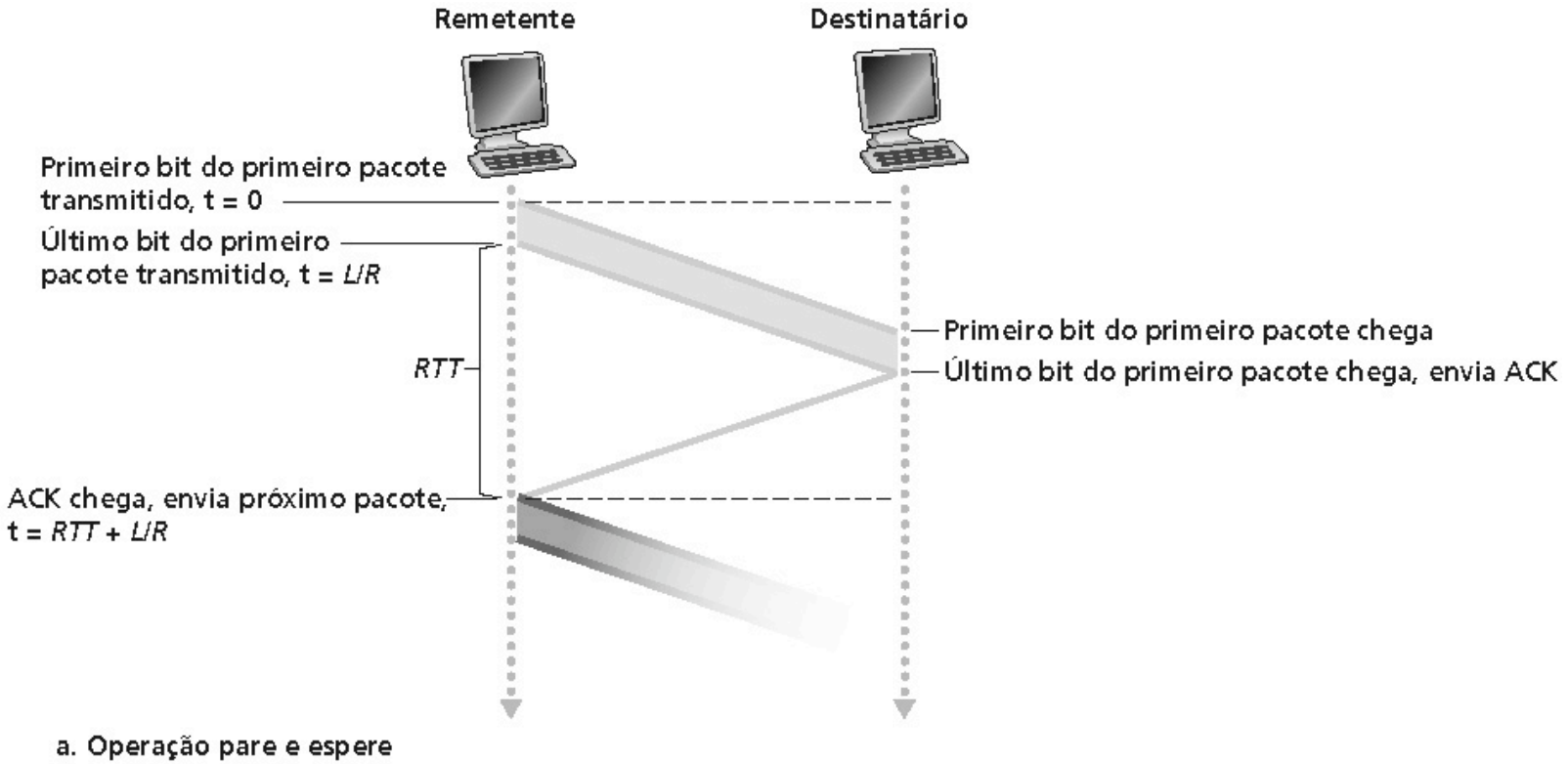
- rdt3.0 funciona, mas o desempenho é sofrível
- Exemplo: enlace de 1 Gbps, 15 ms de atraso de propagação, pacotes de 1 KB:

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

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

- 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*



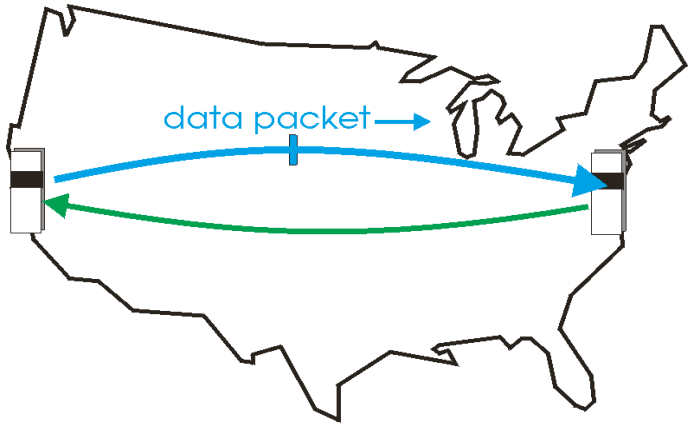
a. Operação pare e espere

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

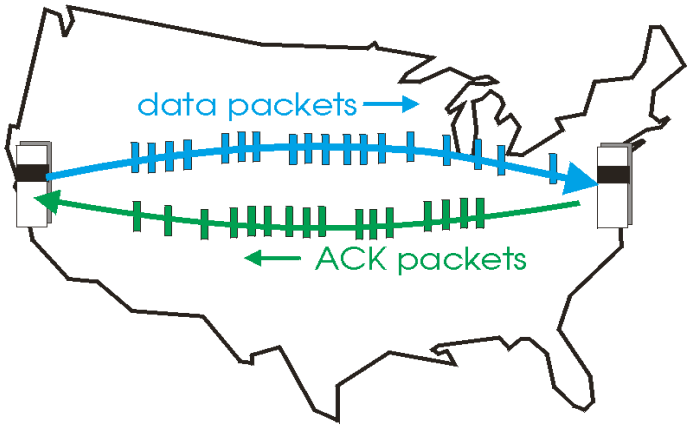
3 Protocolos com paralelismo (pipelining)

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



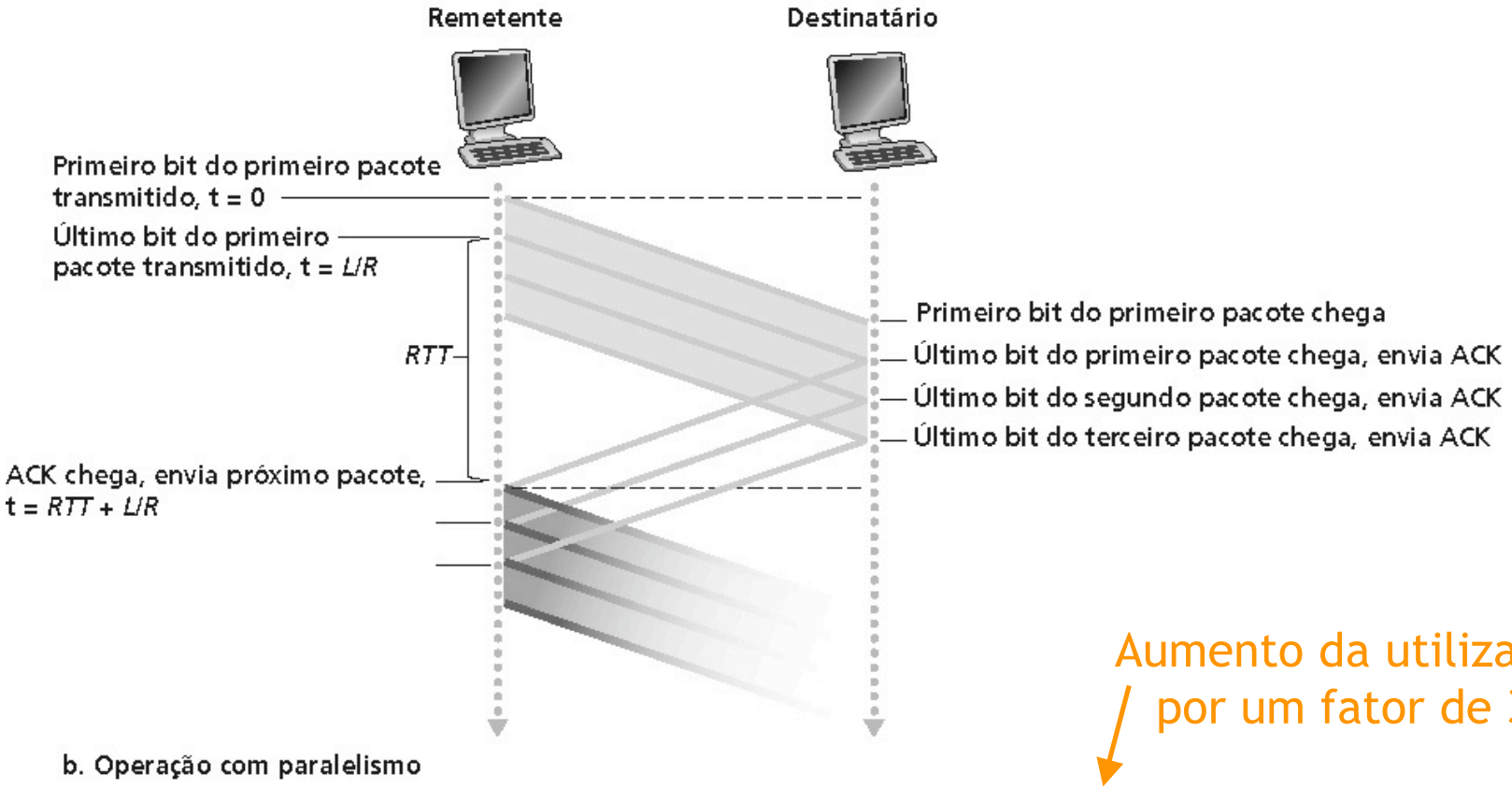
(a) operação do protocolo pare e espere



(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

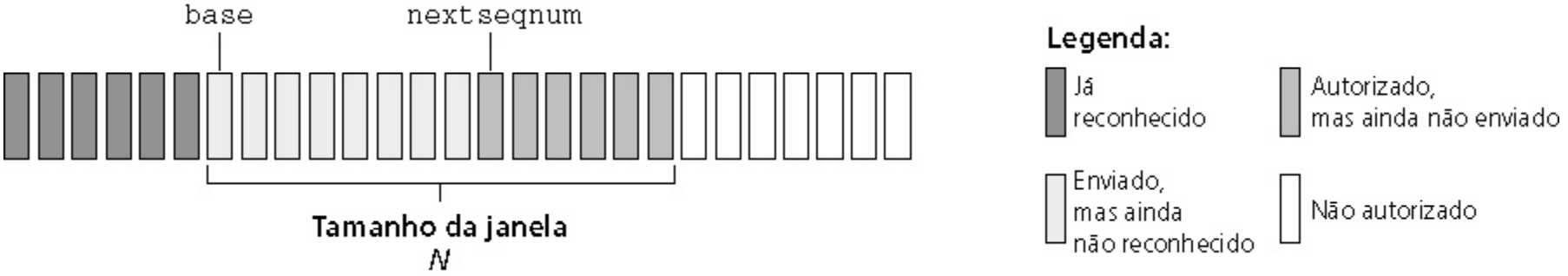
Aumento da utilização por um fator de 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{0{,}024}{30{,}008} = 0{,}0008$$

3 Go-Back-N

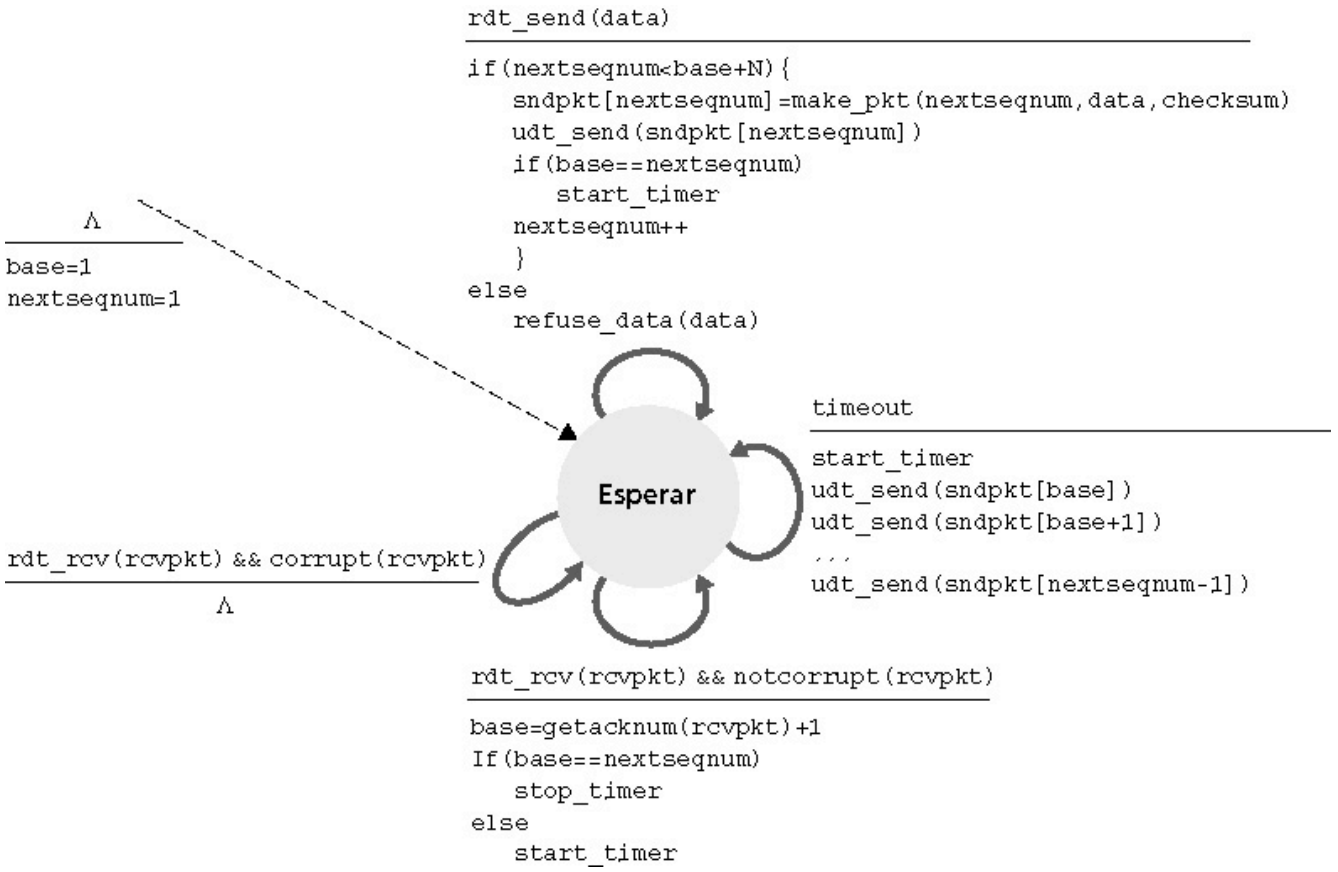
Transmissor:

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

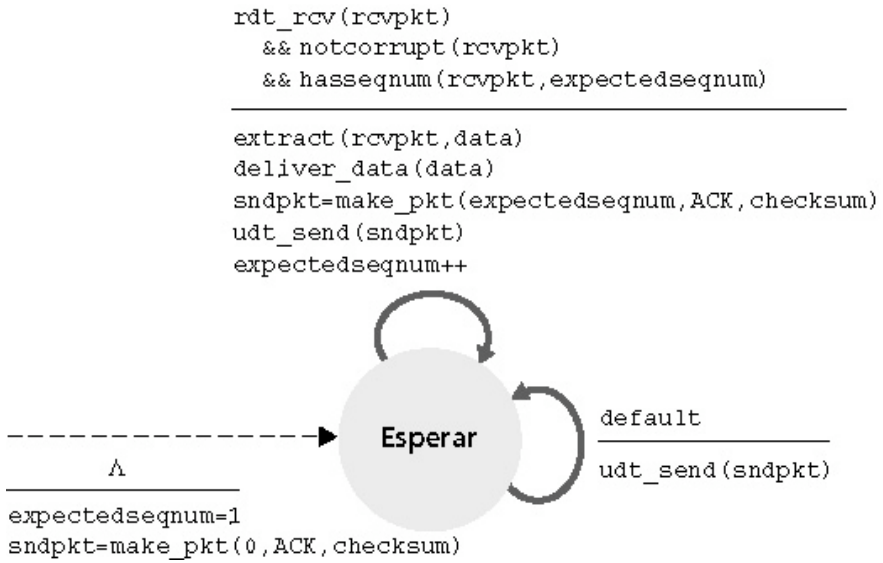


- ACK(n): reconhece todos os pacotes até o número de seqüência N (incluindo este limite). “ACK cumulativo”
  - Pode receber ACKs duplicados (veja receptor)
- Temporizador para cada pacote enviado e não confirmado
- **Tempo de confirmação (n)**: retransmite pacote n e todos os pacotes com número de seqüência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

3 GBN: FSM estendida para o transmissor

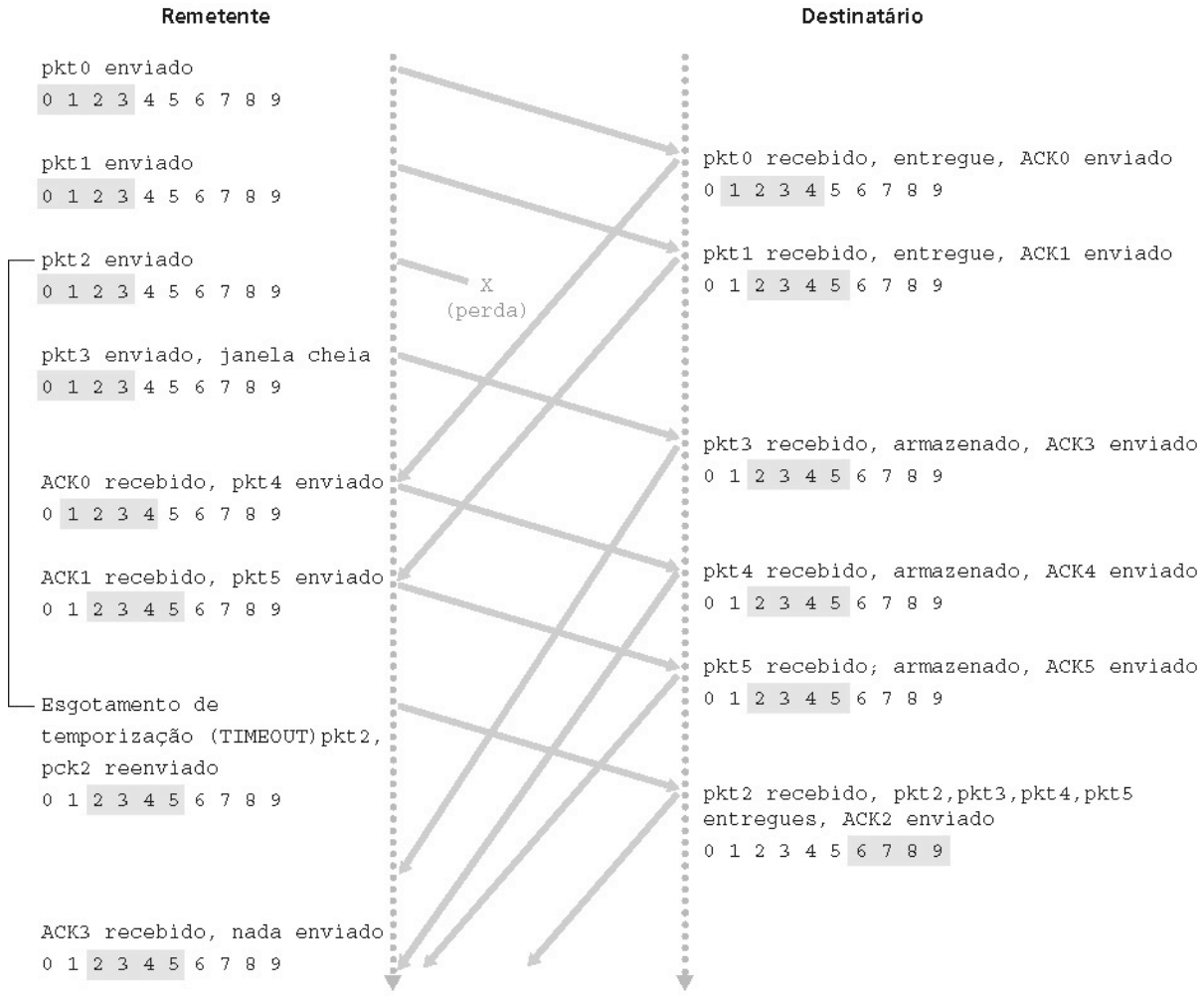


3 GBN: FSM estendida para o receptor



- 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

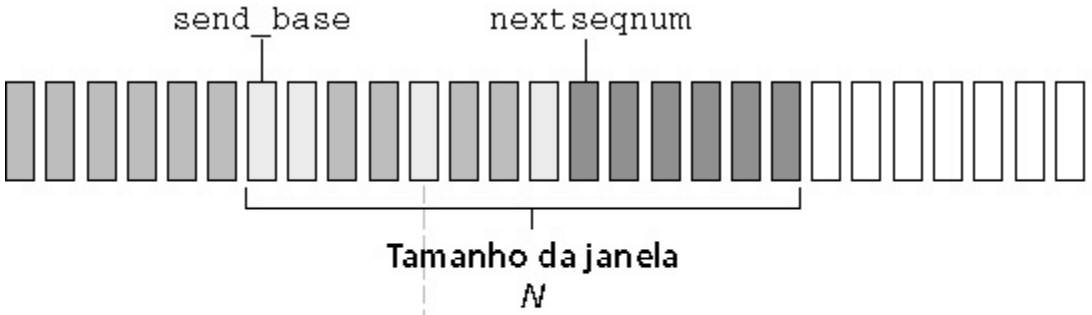
3 GBN em ação



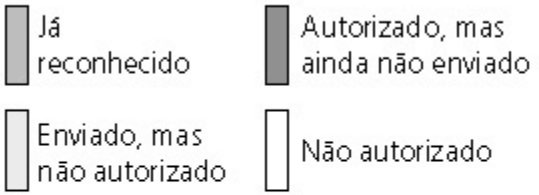
3 Retransmissão seletiva

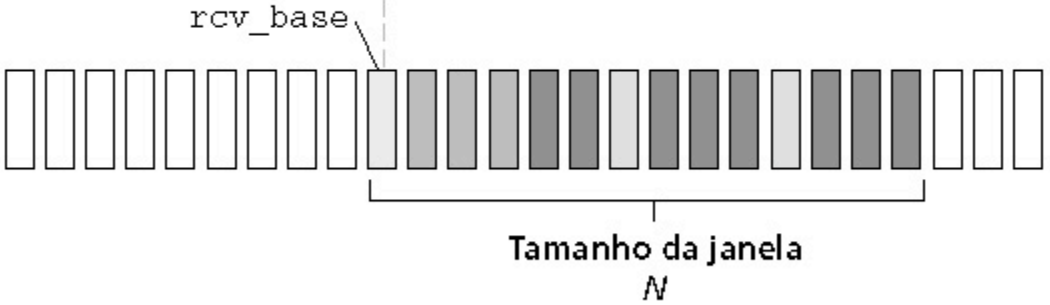
- Receptor reconhece **individualmente** todos os pacotes recebidos corretamente
  - Armazena pacotes, quando necessário, para eventual entrega em ordem para a camada superior
- Transmissor somente reenvia os pacotes para os quais um ACK não foi recebido
  - Transmissor temporiza cada pacote não reconhecido
- 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

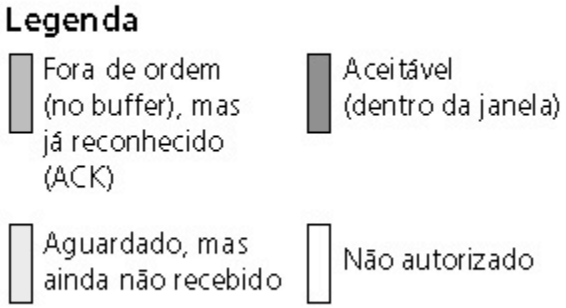


a. Visão que o remetente tem dos números de seqüência





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 próx 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), avan janela para o próximo pacote ainda não recebido

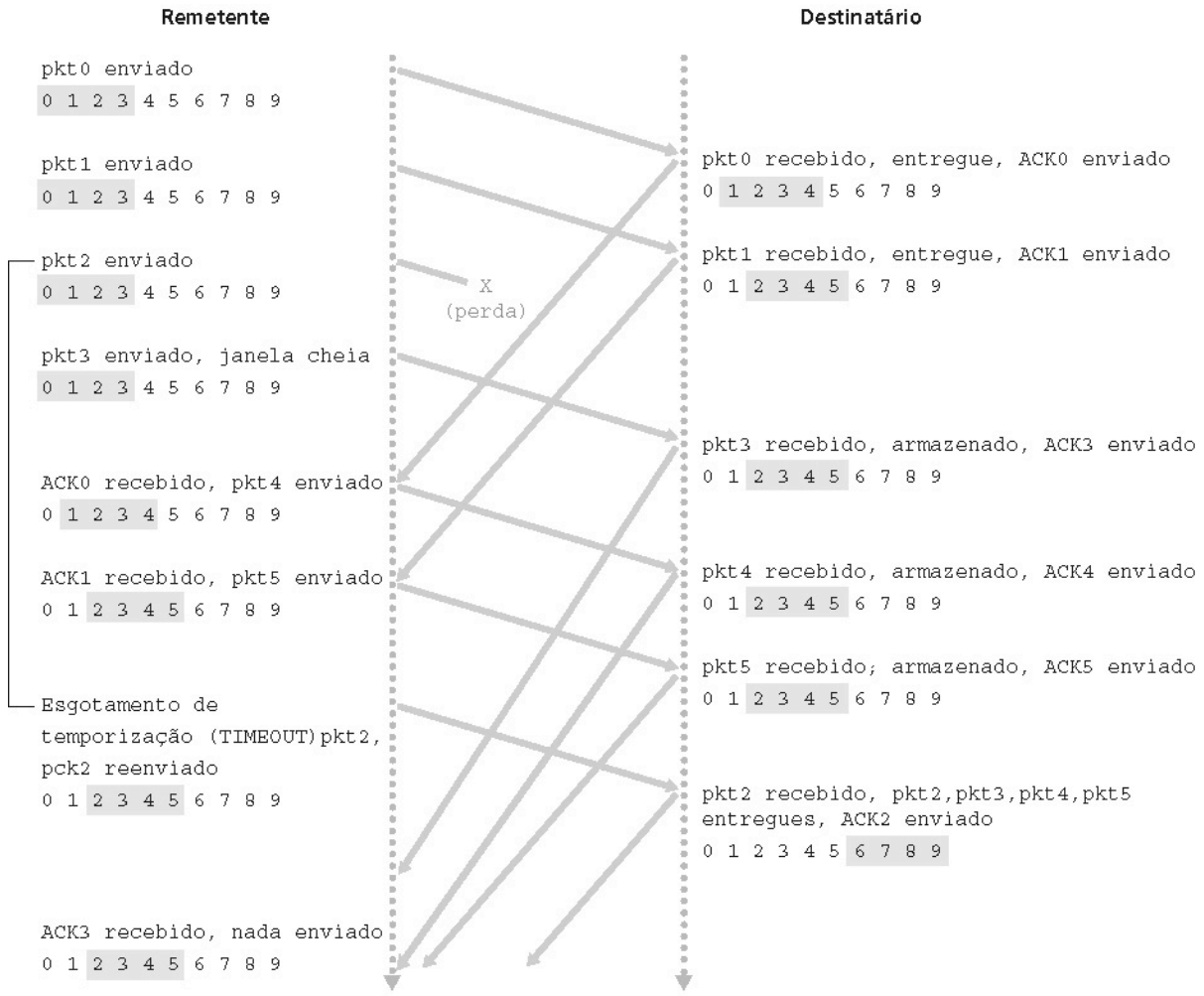
pkt n em [rcvbase-N,rcvbase-1]

- ACK(n)

Caso contrário:

- Ignora

3 Retransmissão seletiva em ação

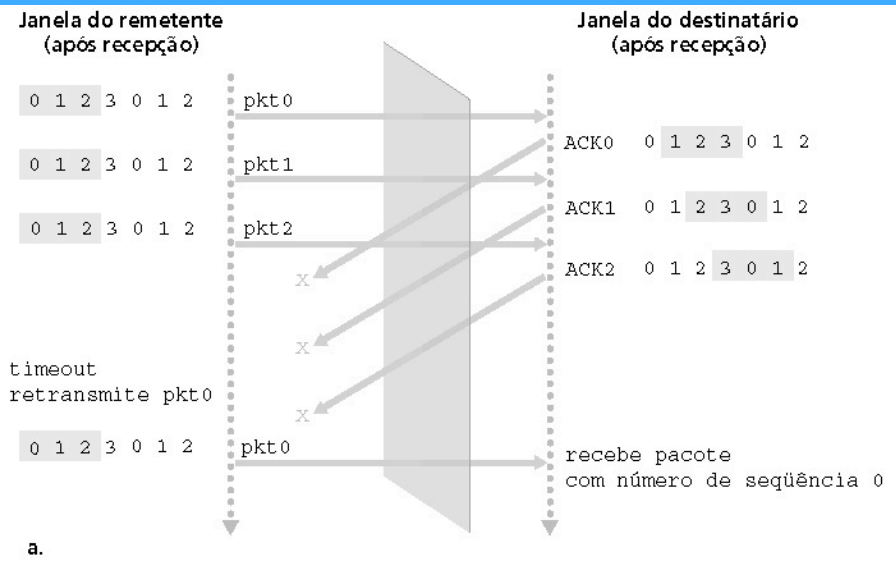


3 Retransmissão seletiva: dilema

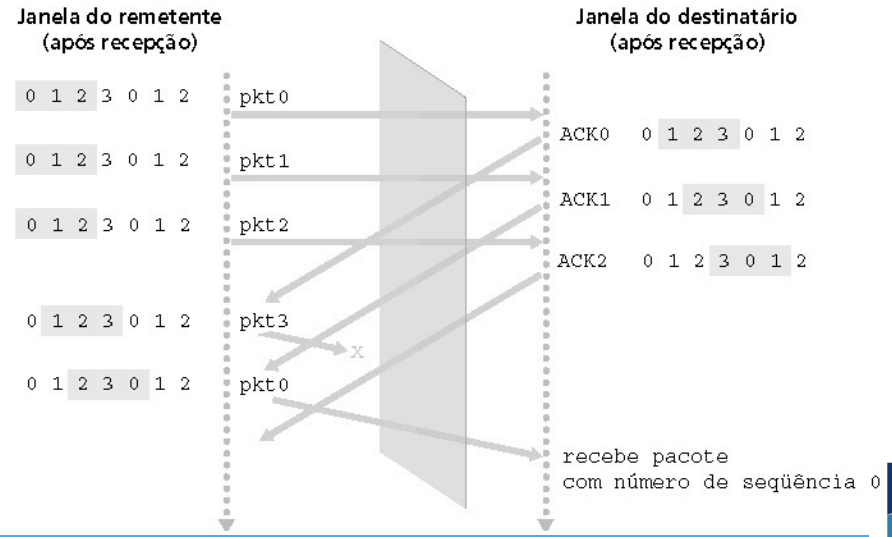
Exemplo:

- 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?



a.



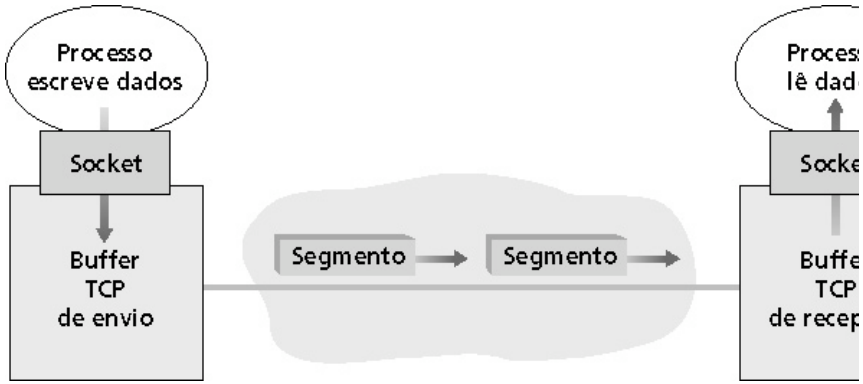
3 Camada de transporte

- 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

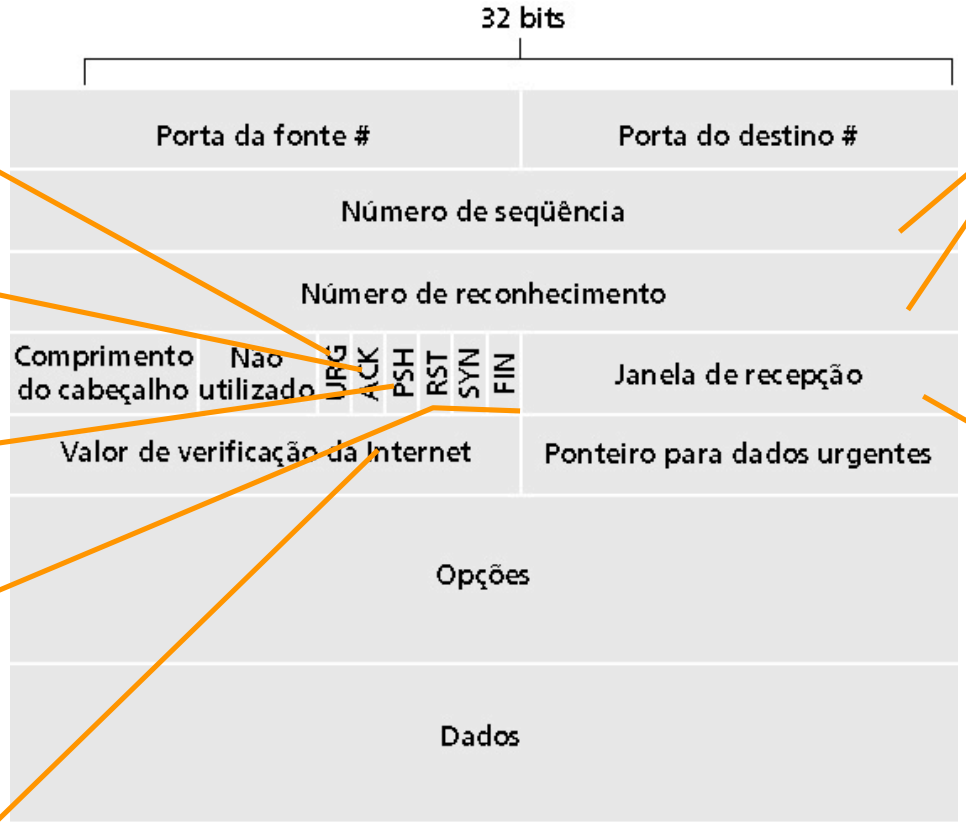
3 TCP: overview

RFCs: 793, 1122, 1323, 2018, 2581

- **Ponto-a-ponto:**
  - Um transmissor, um receptor
- **Confiável, seqüencial byte stream:**
  - Não há contornos de mensagens
- **Pipelined:** (transmissão de vários pacotes sem confirmação)
  - Controle de congestão e de fluxo definem tamanho da janela
- **Buffers de transmissão e de recepção**
- Dados full-duplex:
  - Transmissão bidirecional na mesma conexão
  - MSS: maximum segment size
- Orientado à conexão:
  - Apresentação (troca de mensagens de controle) inicia o estado do transmissor e do receptor antes da troca de dados
- Controle de fluxo:
  - Transmissor não esgota a capacidade do receptor



3 Estrutura do segmento TCP



URG: dados urgentes (pouco usados)

ACK: campo de ACK é válido

PSH: produz envio de dados (pouco usado)

RST, SYN, FIN: estabelec. de conexão (comandos de criação e término)

Internet checksum (como no UDP)

contagem por bytes de dado (não segment

número de byt receptor está pronto para aceitar

3 Número de seqüência e ACKs do TCP

Números de seqüência:

• Número do primeiro byte nos segmentos de dados

ACKs:

• Número do próximo byte esperado do outro lado

•ACK cumulativo

P.: Como o receptor trata segmentos fora de ordem?

• A especificação do TCP não define, fica a critério do implementador