Capítulo 3: Camada de transporte

Objetivos do capítulo:

- entender princípios por trás dos serviços da camada de transporte:
  - multiplexação/demul-tiplexação
  - transferência de dados confiável
  - controle de fluxo
  - controle de congestionamento
- aprender sobre os protocolos da camada de transporte na Internet:
  - UDP: transporte sem conexão
  - TCP: transporte orientado a conexão
  - controle de congestionamento TCP

Capítulo 3: Esboço

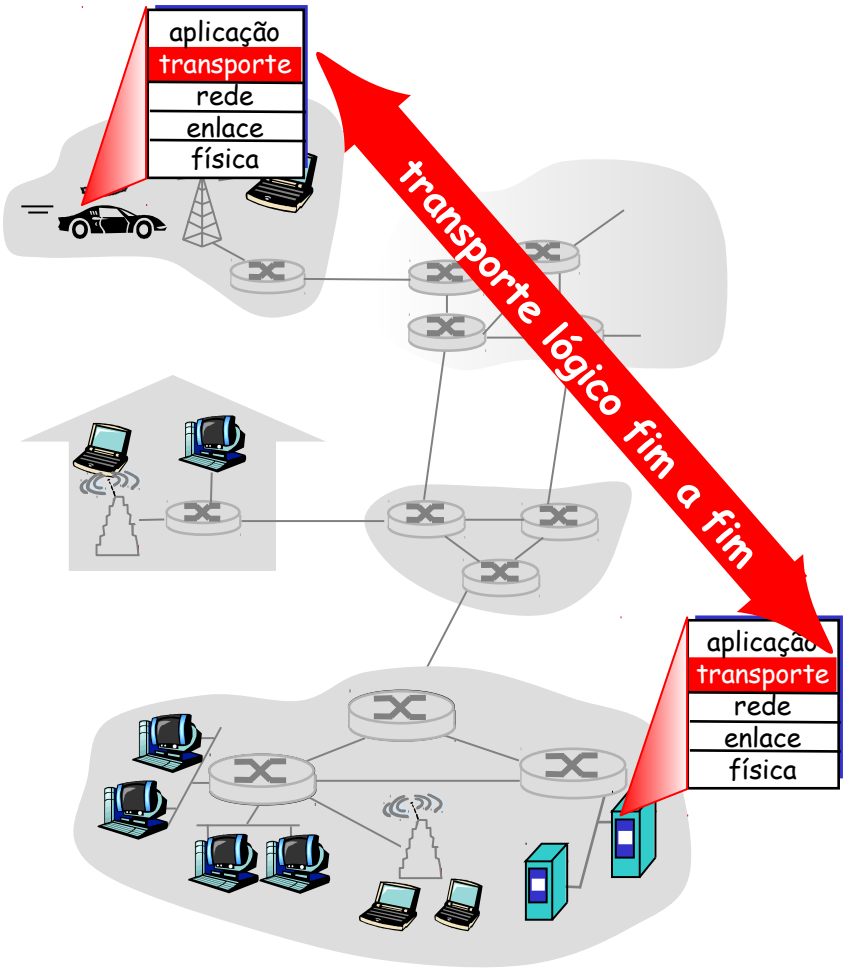
- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

Resumo

Camada	Problema	Serviço de Comunicação	Protocolos
Aplicacação			HTTP, DNS, openSVN DHT, SSL
Transporte	Confiabilidade Congestionamento	Entre processos	TPC, UDP
Rede	Endereçamento Roteamento	Entre sistemas	IP, ICMP, RIP, OSPF
Enlace	Codificação Compartilhamento	Entre vizinhos	Ethernet, Wi-fi

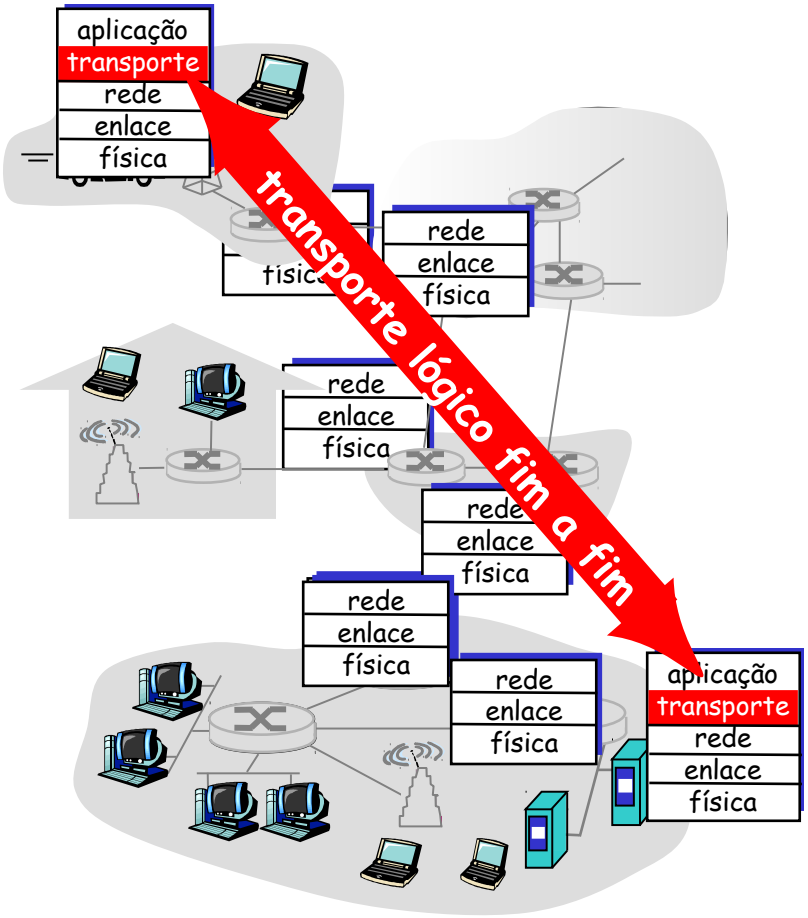
Serviços e protocolos de transporte

- oferecem *comunicação lógica* entre processos de aplicação rodando em hospedeiros diferentes
- protocolos de transporte rodam em sistemas finais
  - lado remetente: divide as msgs da aplicação em segmentos, passa à camada de rede
  - lado destinatário: remonta os segmentos em msgs, passa à camada de aplicação
- mais de um protocolo de transporte disponível às aplicações
  - Internet: TCP e UDP



Protocolos da camada de transporte da Internet

- remessa confiável e em ordem (TCP)
  - controle de congestionamento
  - controle de fluxo
  - estabelecimento da conexão
- remessa não confiável e desordenada: UDP
  - extensão sem luxo do IP pelo "melhor esforço"
- serviços não disponíveis:
  - garantias de atraso
  - garantias de largura de banda



Capítulo 3: Esboço

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

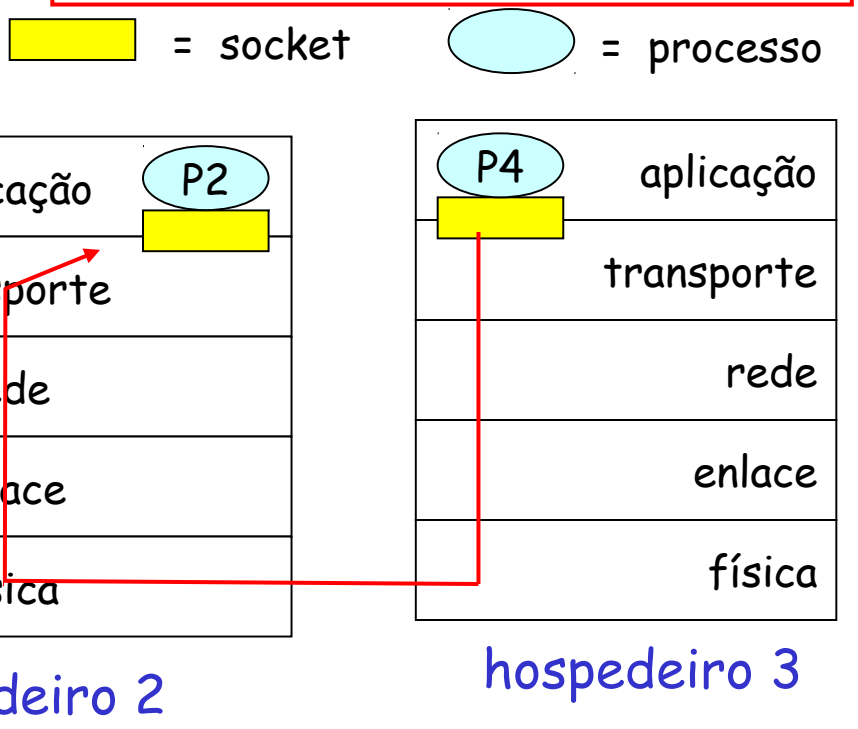
Multiplexação/ demultiplexação

multiplexação no remetente:

colhendo dados de múltiplos sockets, envelopando dados com cabeçalho (usados depois para demultiplexação)

demultiplexação no destinatário:

entregando segmentos recebidos ao socket correto



Como funciona a demultiplexação

- hospedeiro recebe datagramas IP
  - cada pacote tem endereço IP de origem, endereço IP de destino
  - cada pacote carrega 1 segmento da camada de transporte
  - cada segmento tem número de porta de origem, destino
- hospedeiro usa endereços IP & números de porta para direcionar segmento ao socket apropriado

formato do segmento TCP/UDP

Demultiplexação não orientada para conexão

- cria sockets com números de porta:

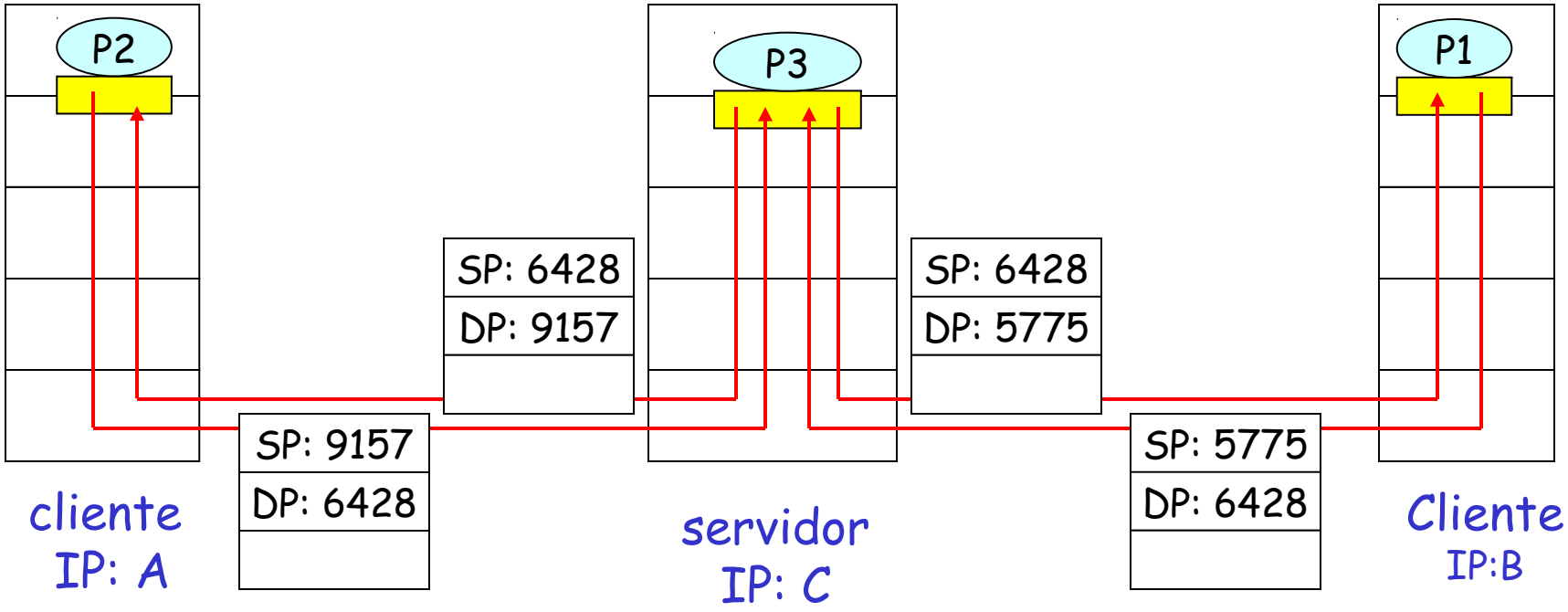
```
DatagramSocket mySocket1 = new
   DatagramSocket(12534);
DatagramSocket mySocket2 = new
   DatagramSocket(12535);
```

- socket UDP identificado por tupla de dois elementos:

(endereço IP destino, número porta destino)

- quando hospedeiro recebe segmento UDP:
  - verifica número de porta de destino no segmento
  - direciona segmento UDP para socket com esse número de porta
- datagramas IP com diferentes endereços IP de origem e/ou números de porta de origem direcionados para o mesmo socket

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

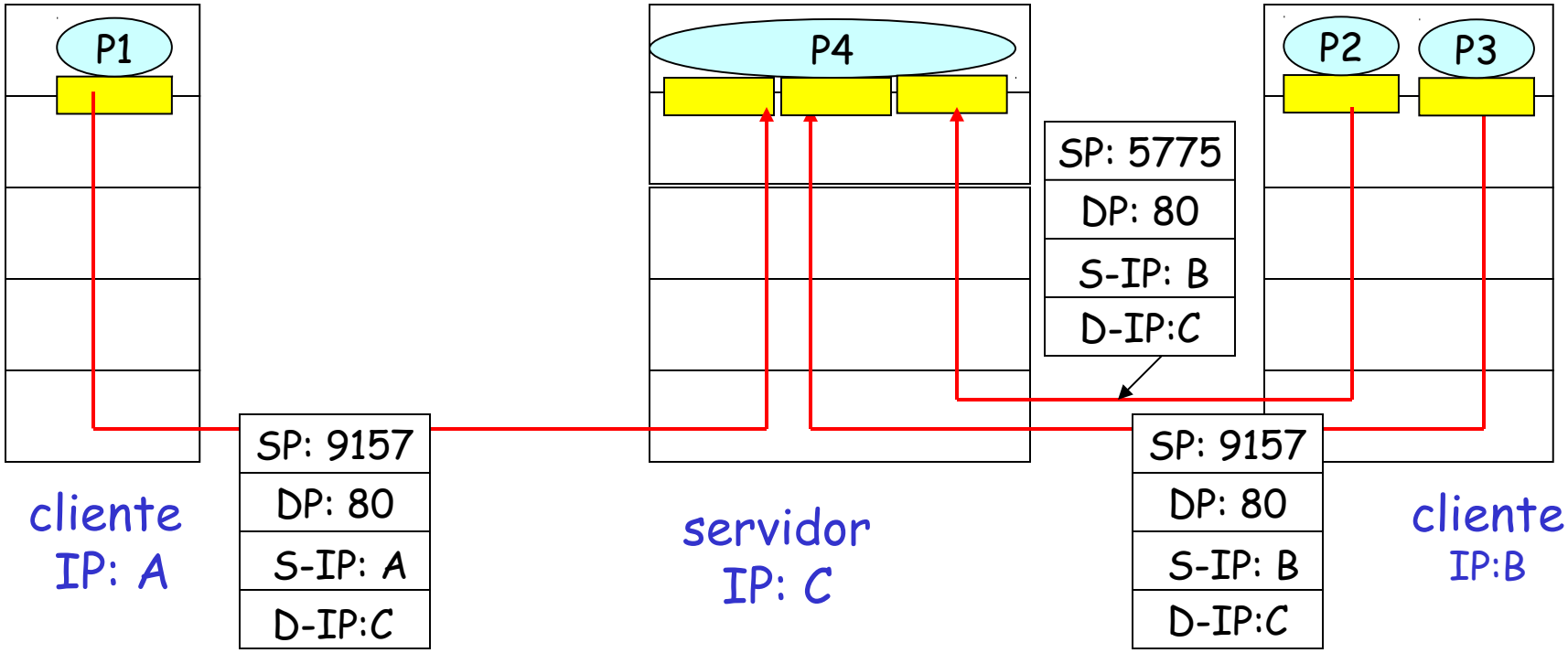


SP oferece "endereço de retorno"

Demultiplexação orientada para conexão

- socket TCP identificado por tupla de 4 elementos:
  - endereço IP de origem
  - número de porta de origem
  - endereço IP de destino
  - número de porta de destino
- hospedeiro destinatário usa todos os quatro valores para direcionar segmento ao socket apropriado
- hospedeiro servidor pode admitir muitos sockets TCP simultâneos:
  - cada socket identificado por sua própria tupla de 4
- servidores Web têm diferentes sockets para cada cliente conectando
  - HTTP não persistente terá diferentes sockets para cada requisição

Demultiplexação orientada para conexão: servidor Web threaded



Capítulo 3: Esboço

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

UDP: User Datagram Protocol [RFC 768]

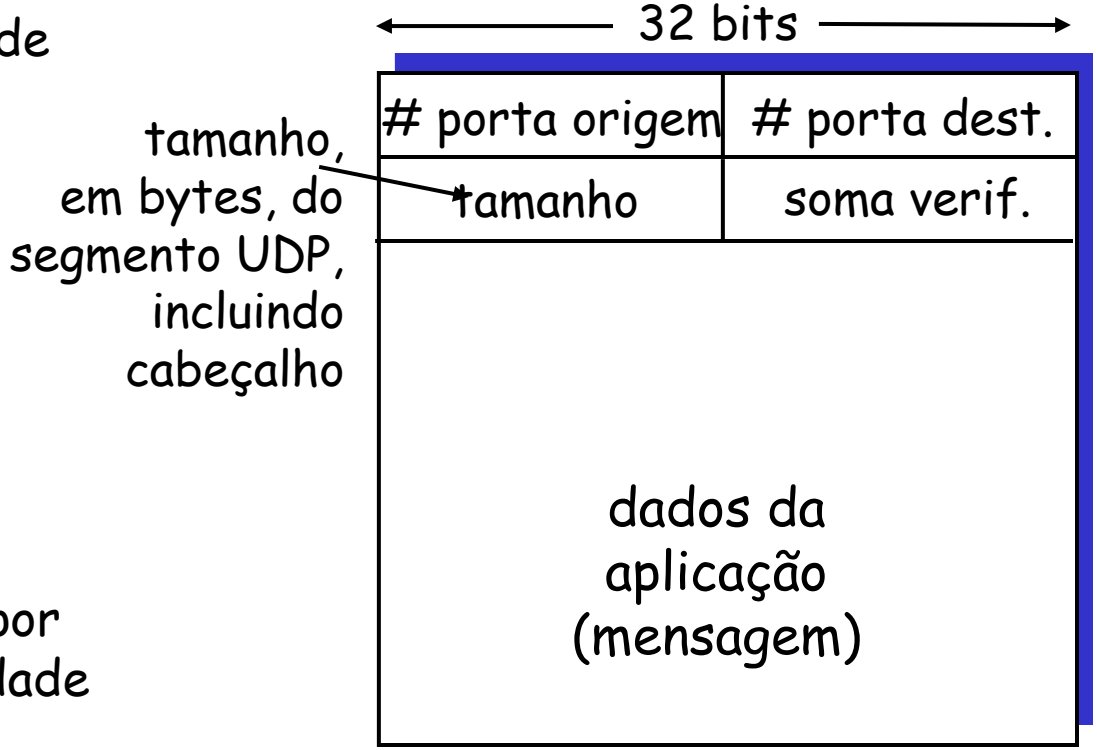
- protocolo de transporte da Internet "sem luxo", básico
- serviço de "melhor esforço", segmentos UDP podem ser:
  - perdidos
  - entregues à aplicação fora da ordem
- *sem conexão:*
  - sem handshaking entre remetente e destinatário UDP
  - cada segmento UDP tratado independente dos outros

Por que existe um UDP?

- sem estabelecimento de conexão (que pode gerar atraso)
- simples: sem estado de conexão no remetente, destinatário
- cabeçalho de segmento pequeno
- sem controle de congestionamento: UDP pode transmitir o mais rápido possível

UDP

- normalmente usado para streaming de aplicações de multimídia
  - tolerante a perdas
  - sensível à taxa
- outros usos do UDP
  - DNS
  - RIP
  - DHCP
- transferência confiável por UDP: aumenta confiabilidade na camada de aplicação
  - recuperação de erro específica da aplicação!



formato de segmento UDP

Capítulo 3: Esboço

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado para conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento no TCP

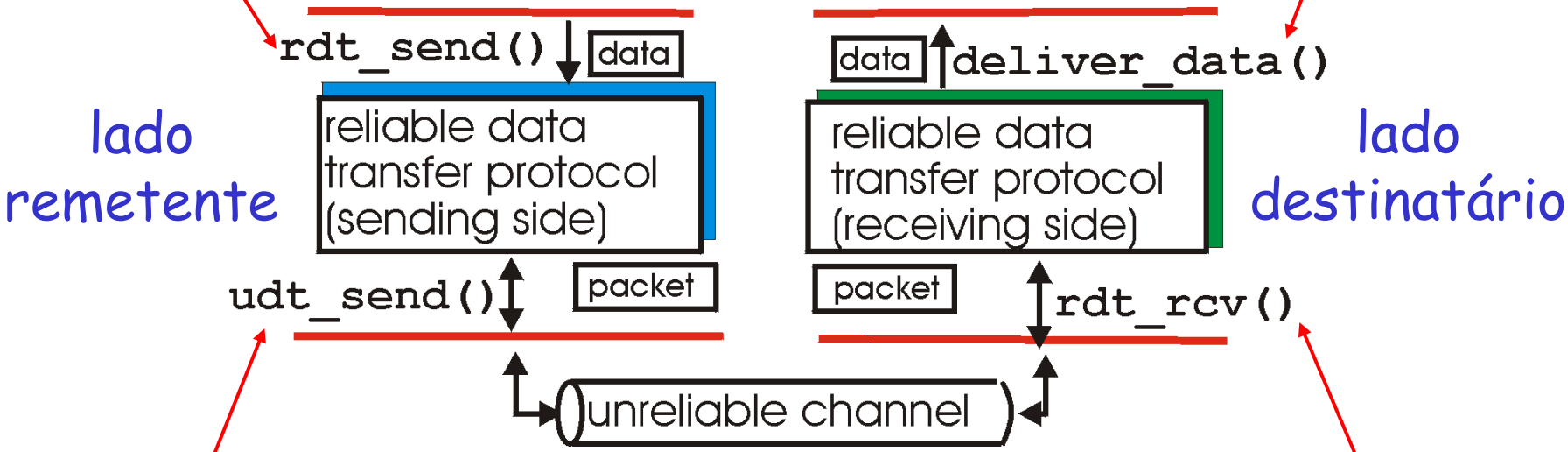
Princípios de transferência confiável de dados

- importante nas camadas de aplicação, transporte e enlace
- um dos mais importantes tópicos de redes!
- características do canal confiável determinarão complexidade do protocolo de transferência confiável (rdt-reliable data transfer)

Transferência confiável de dados: introdução

rdt_send(): chamado de cima, (p. e., pela apl.). Dados passados para remeter à camada superior do destinatário

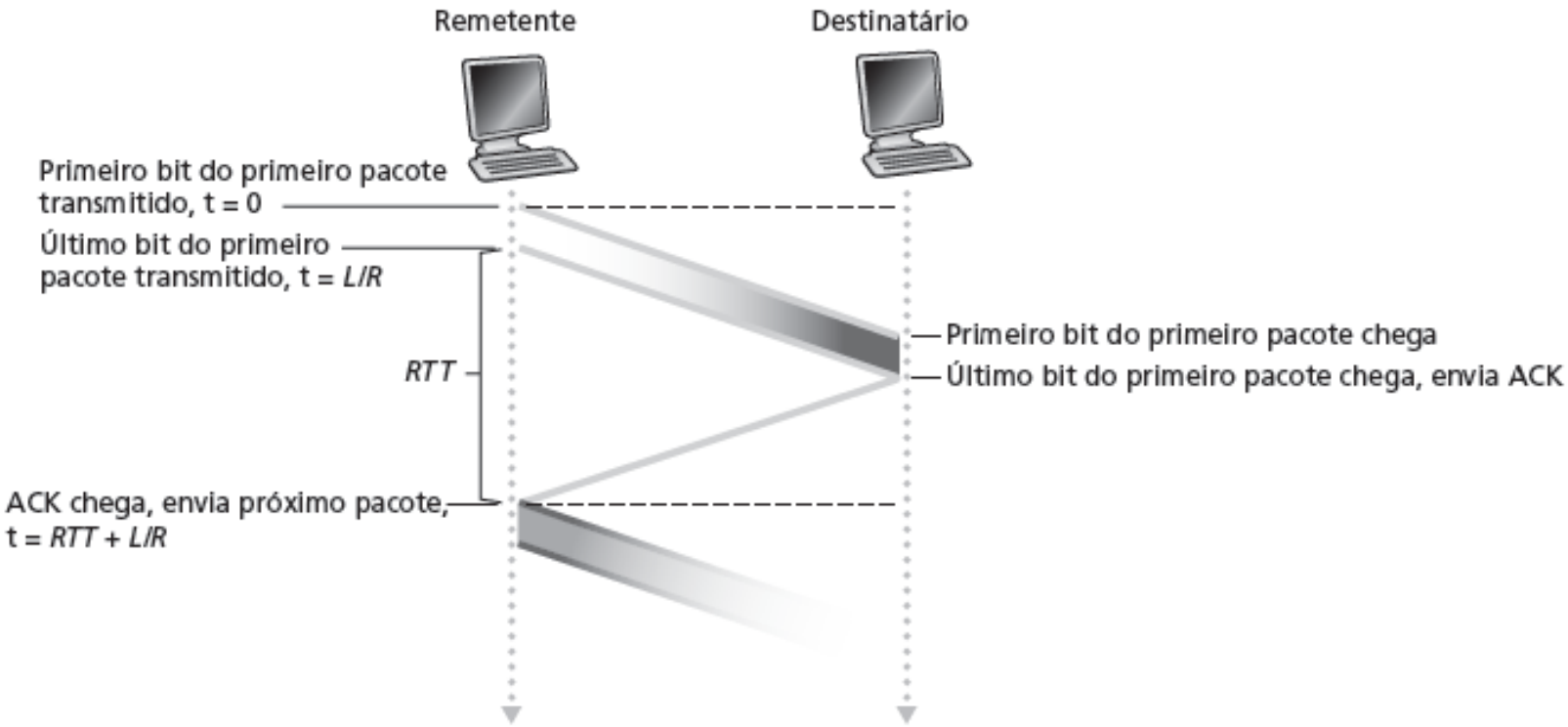
deliver_data(): chamado pela rdt para remeter dados para cima



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

rdt3.0: operação pare e espere

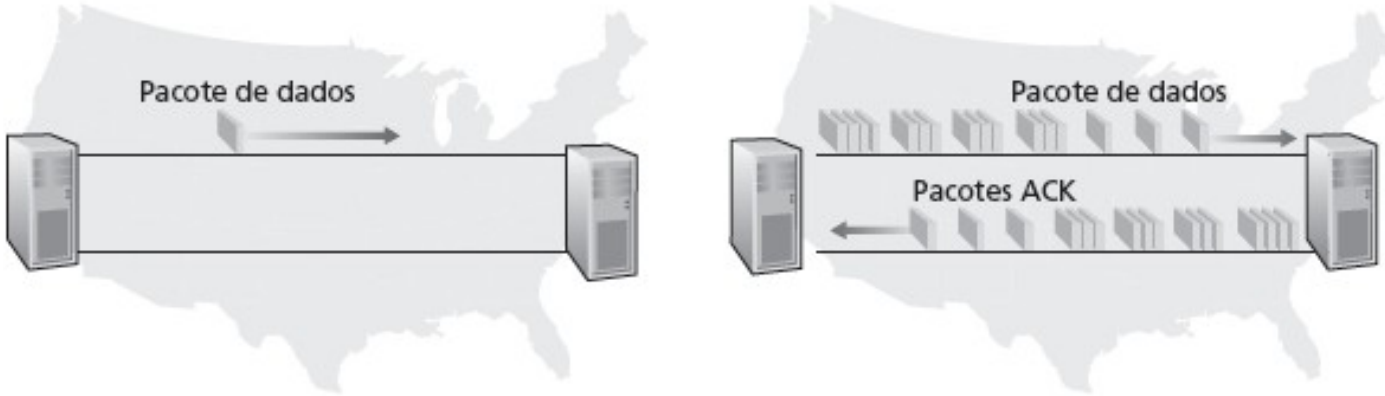


$$U_{remet} = \frac{L/R}{RTT + L/R}$$

Protocolos com paralelismo

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

- deve-se utilizar um intervalo de números de sequência
- buffering no remetente e/ou destinatário

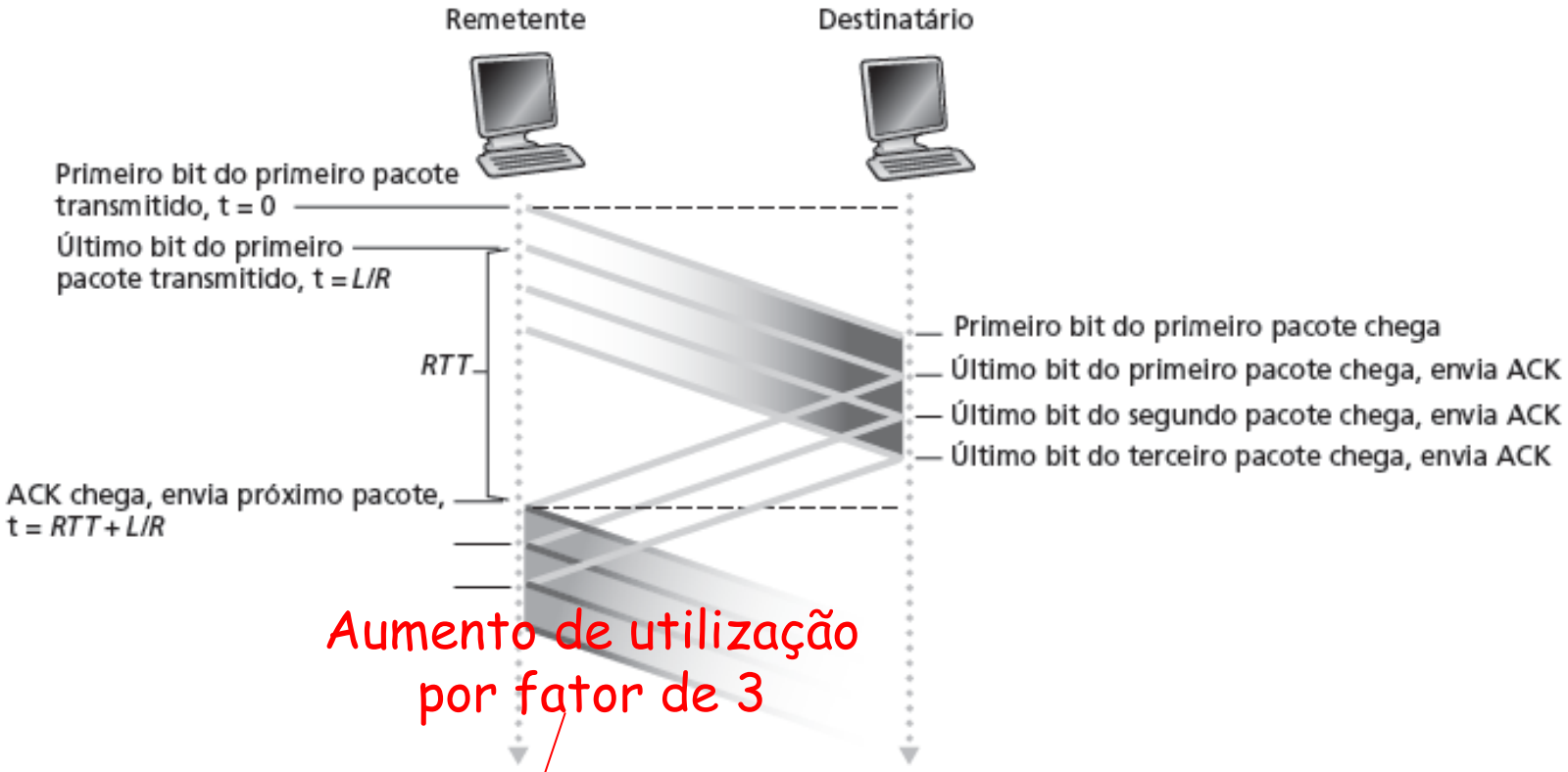


a. Um protocolo pare e espere em operação

b. Um protocolo com paralelismo em operação

- duas formas genéricas de protocolo com paralelismo: *Go-Back-N, repetição seletiva*

Paralelismo: utilização aumentada



$$U_{remet} = \frac{3 \times L/R}{RTT + L/R}$$

Protocolos com paralelismo

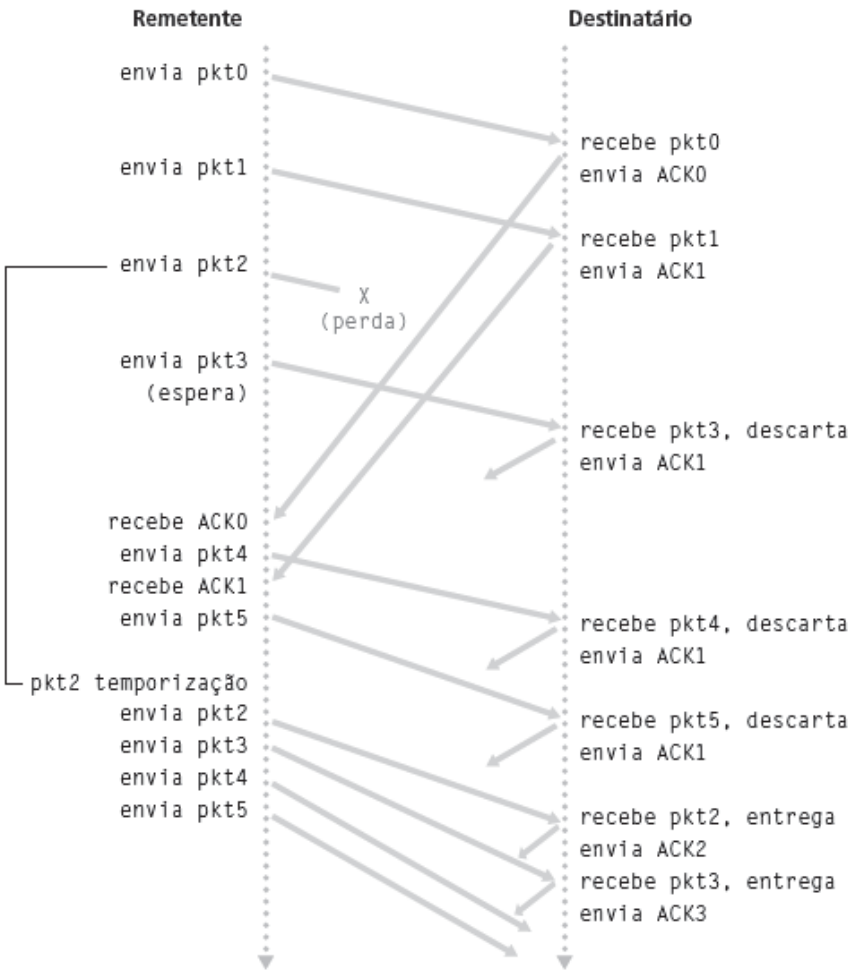
Go-back-N: visão geral

- *remetente:* até N pacotes não reconhecidos no pipeline
- *destinatário:* só envia ACKs cumulativos
  - não envia pct ACK se houver uma lacuna
- *remetente:* tem temporizador para pct 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 pct sem ACK
  - se o temporizador expirar: retransmite apenas o pacote sem ACK

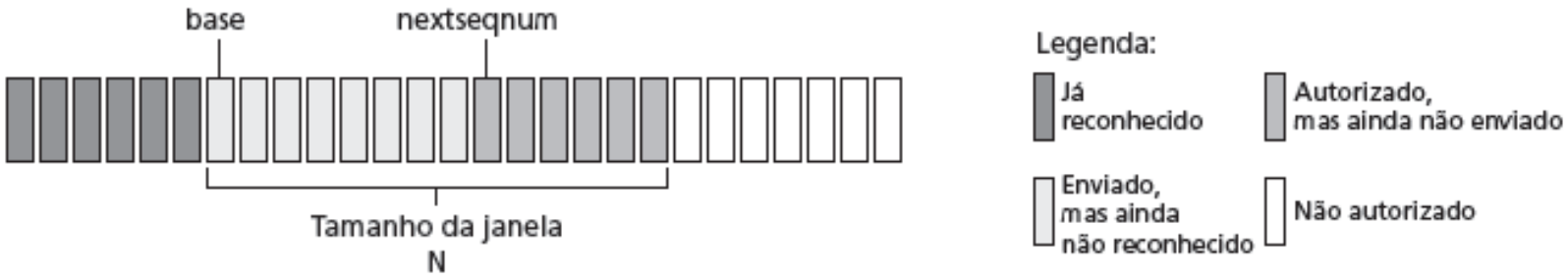
GBN em operação - N=4



Go-Back-N

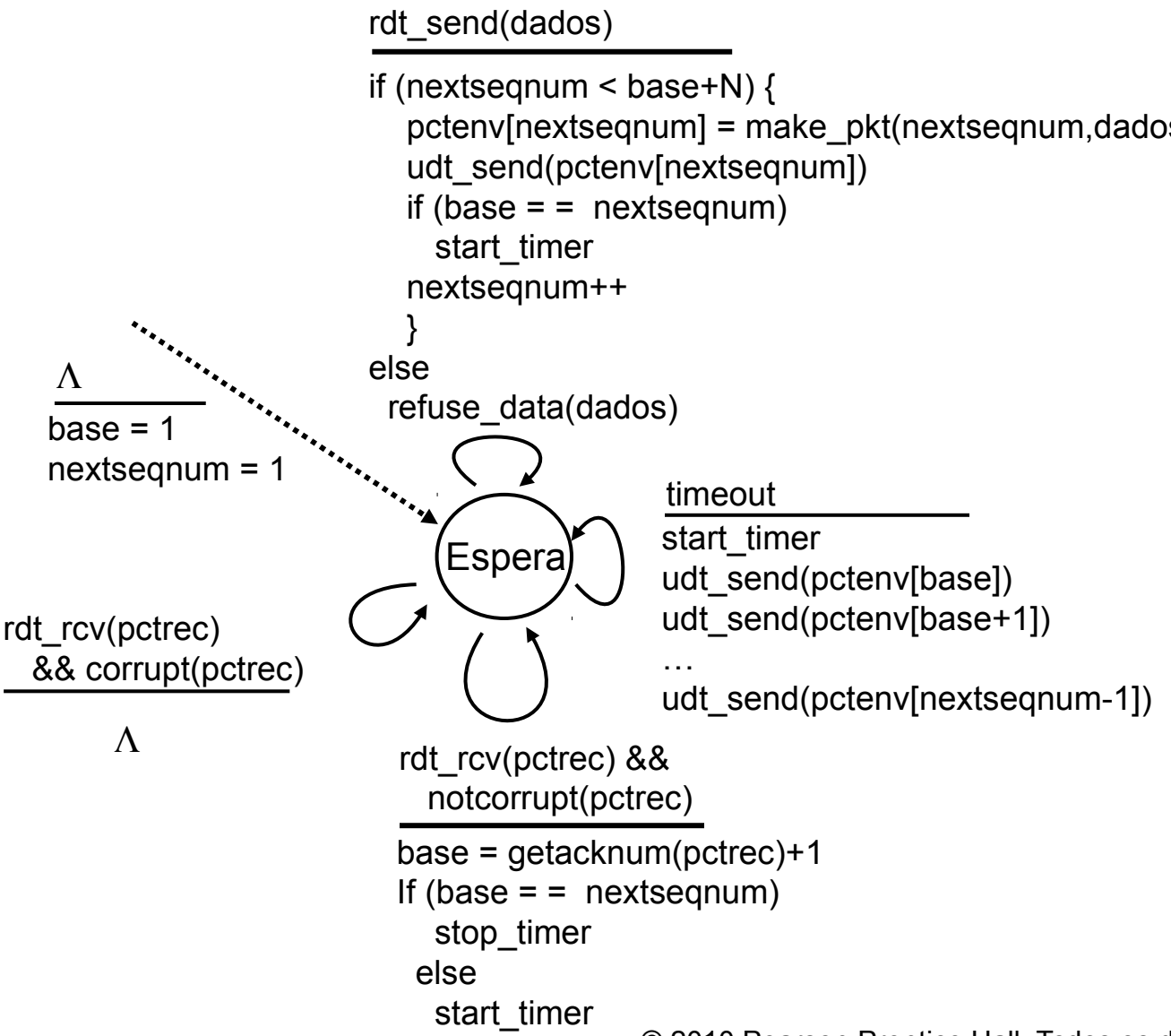
remetente:

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

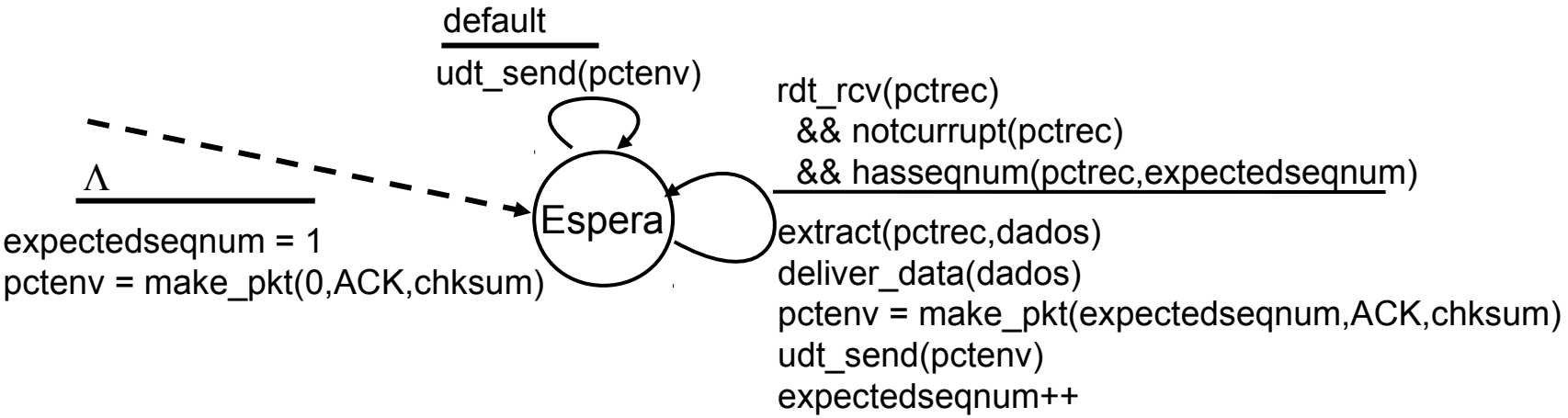


- ACK(n): ACK de todos pcts até inclusive # seq. n – "ACK cumulativo"
  - pode receber ACKs duplicados (ver destinatário)
- temporizador para o pacote mais antigo no ar
- *timeout(n):* retransmite pct n e todos pcts com # seq. mais alto na janela

GBN: FSM no remetente



GBN: FSM no destinatário



apenas ACK: sempre envia ACK para pct recebido corretamente com # seq. mais alto *em ordem*

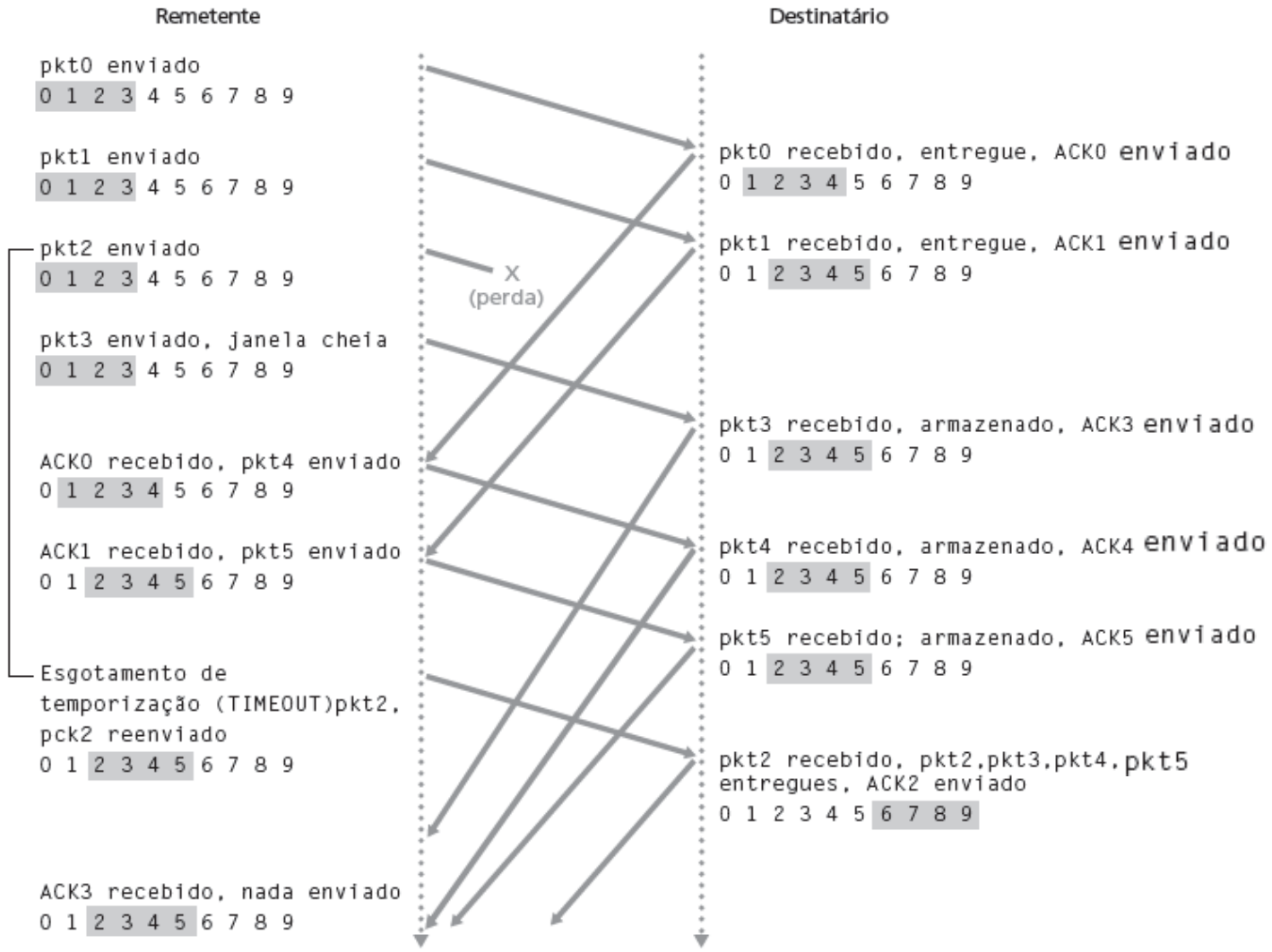
- pode gerar ACKs duplicados
- só precisa se lembrar de **`expectedseqnum`**

- pacote fora de ordem:
  - descarta (não mantém em buffer) -> sem buffering no destinatário!
  - reenvia ACK do pct com # seq. mais alto em ordem

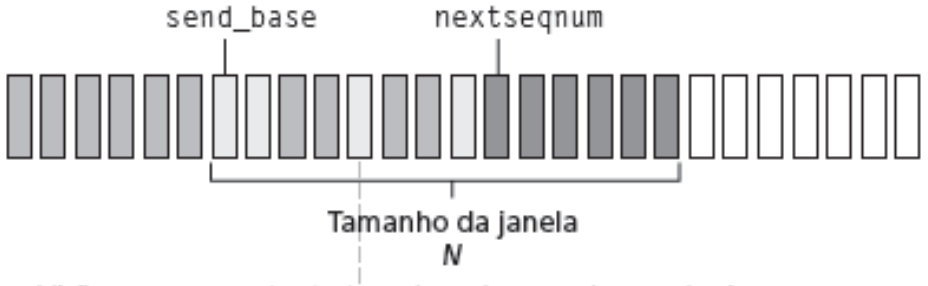
Repetição seletiva

- destinatário reconhece *individualmente* todos os pacotes recebidos de modo correto
  - mantém pcts em buffer, se for preciso, para eventual remessa em ordem para a camada superior
- remetente só reenvia pcts para os quais o ACK não foi recebido
  - temporizador no remetente para cada pct sem ACK
- janela do remetente
  - N # seq. consecutivos
  - novamente limita #s seq. de pcts enviados, sem ACK

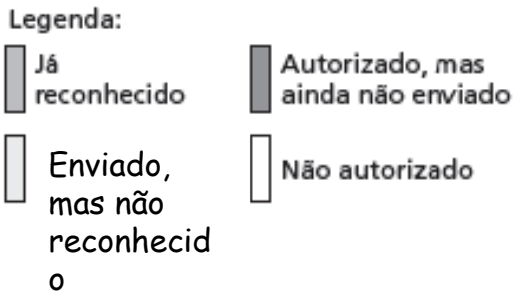
Repetição seletiva em operação

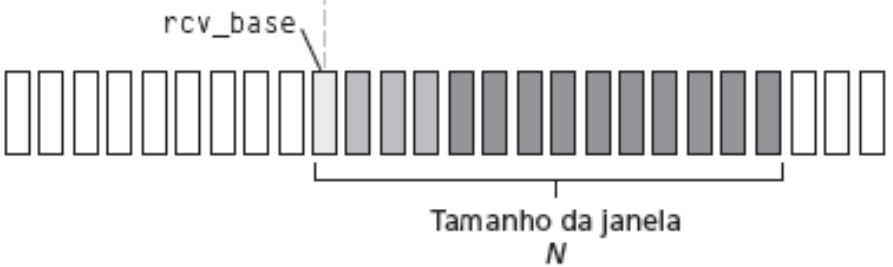


Repetição seletiva: janelas de remetente, destinatário

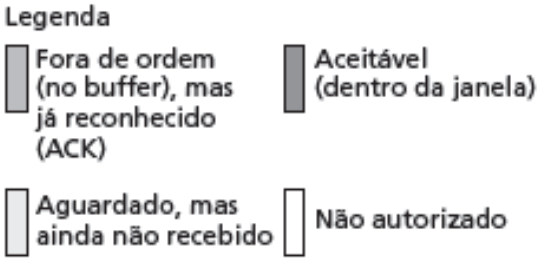


a. Visão que o remetente tem dos números de sequência





b. Visão que o destinatário tem dos números de sequência



Repetição seletiva

remetente

dados de cima:

- se próx. # seq. disponível na janela, envia pct

***timeout*(n):**

- reenvia pct n, reinicia temporizador

ACK(n) em [sendbase,sendbase+N]:

- marca pct n como recebido
- se n menor pct com ACK, avança base da janela para próximo # seq. sem ACK

destinatário

pct n em [rcvbase, rcvbase+N-1]

- envia ACK(n)
- fora de ordem: buffer
- em ordem: entrega (também entrega pcts em ordem no buffer), avança janela para próximo pct ainda não recebido

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

- ACK(n)

caso contrário:

- ignora