

Parte 1

Camada de Transporte

1 Fundamentos

- Funções da Camada de Transporte
- Relações entre as Camadas de Transporte e Rede
- A Camada de Transporte na Internet
- Multiplexação e Demultiplexação
- Segmentação e Reagrupamento
- Endereçamento de Porta

2 Transporte sem Conexão: UDP

3 Princípios da Transferência Confiável de Dados

4 Transporte Orientado à Conexão: TCP

5 Controle de Congestionamento

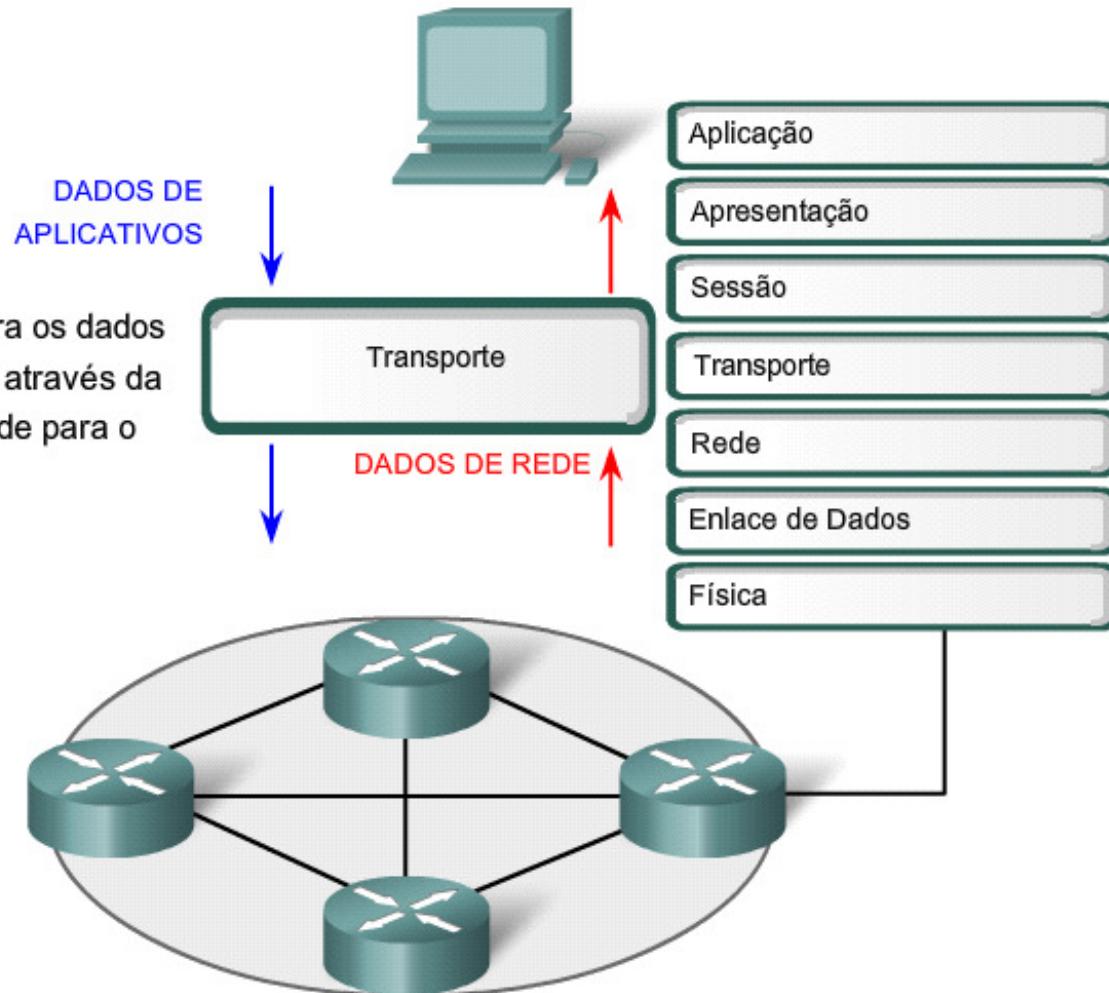
Fundamentos

- Cada uma das aplicações da Internet são **empacotadas, transportadas e entregues** ao servidor daemon apropriado ou aplicação no dispositivo de destino;
- Os processos descritos na camada de Transporte aceitam dados da camada de Aplicação e os preparam para endereçamento na camada de Rede;
- A camada de Transporte é responsável pela **transferência fim-a-fim** geral de dados de aplicação;
- Os pacotes da camada de Transporte são denominados **segmentos**.

Fundamentos

A Camada de Transporte OSI

A camada de Transporte prepara os dados de aplicativos para o transporte através da rede e processa os dados da rede para o uso pelos aplicativos.



Conteúdo

1 Fundamentos

- Funções da Camada de Transporte
- Relações entre as Camadas de Transporte e Rede
- A Camada de Transporte na Internet
- Multiplexação e Demultiplexação
- Segmentação e Reagrupamento
- Endereçamento de Porta

2 Transporte sem Conexão: UDP

3 Princípios da Transferência Confiável de Dados

4 Transporte Orientado à Conexão: TCP

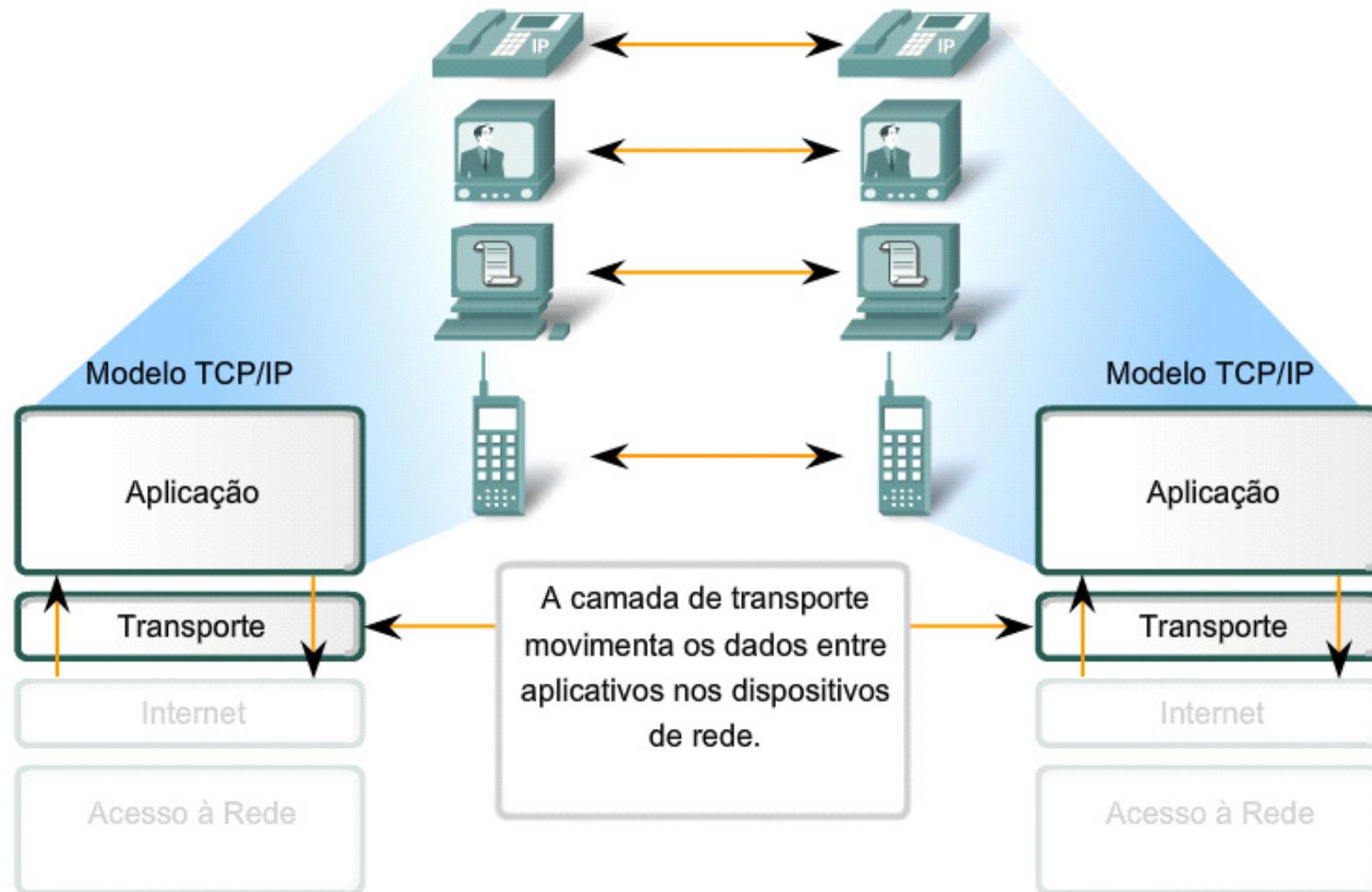
5 Controle de Congestionamento

Funções da Camada de Transporte

- ① A camada de Transporte proporciona a **segmentação de dados** e o controle necessário para **reagrupar esses segmentos** em fluxos de comunicação:
 - **Rastrear a comunicação individual** entre as aplicações nos hosts de origem e destino;
 - **Identificar** as diferentes aplicações;
 - **Segmentar dados** e gerenciar cada segmento;
 - **Reagrupar os segmentos** em fluxos de dados de aplicação.
- ② Assegura que, se necessário, todos os dados sejam **recebidos confiavelmente e em ordem** pela aplicação correta;
- ③ Emprega mecanismos de **tratamento de erros**: campos de detecção de erros nos cabeçalhos dos segmentos;
- ④ Provê **Controle de Congestionamento**: serviço dirigido para o bem geral da Internet.

Funções da Camada de Transporte

Habilitando Aplicativos em Dispositivos para a Comunicação



Funções da Camada de Transporte

● Rastreamento de Conversações Individuais

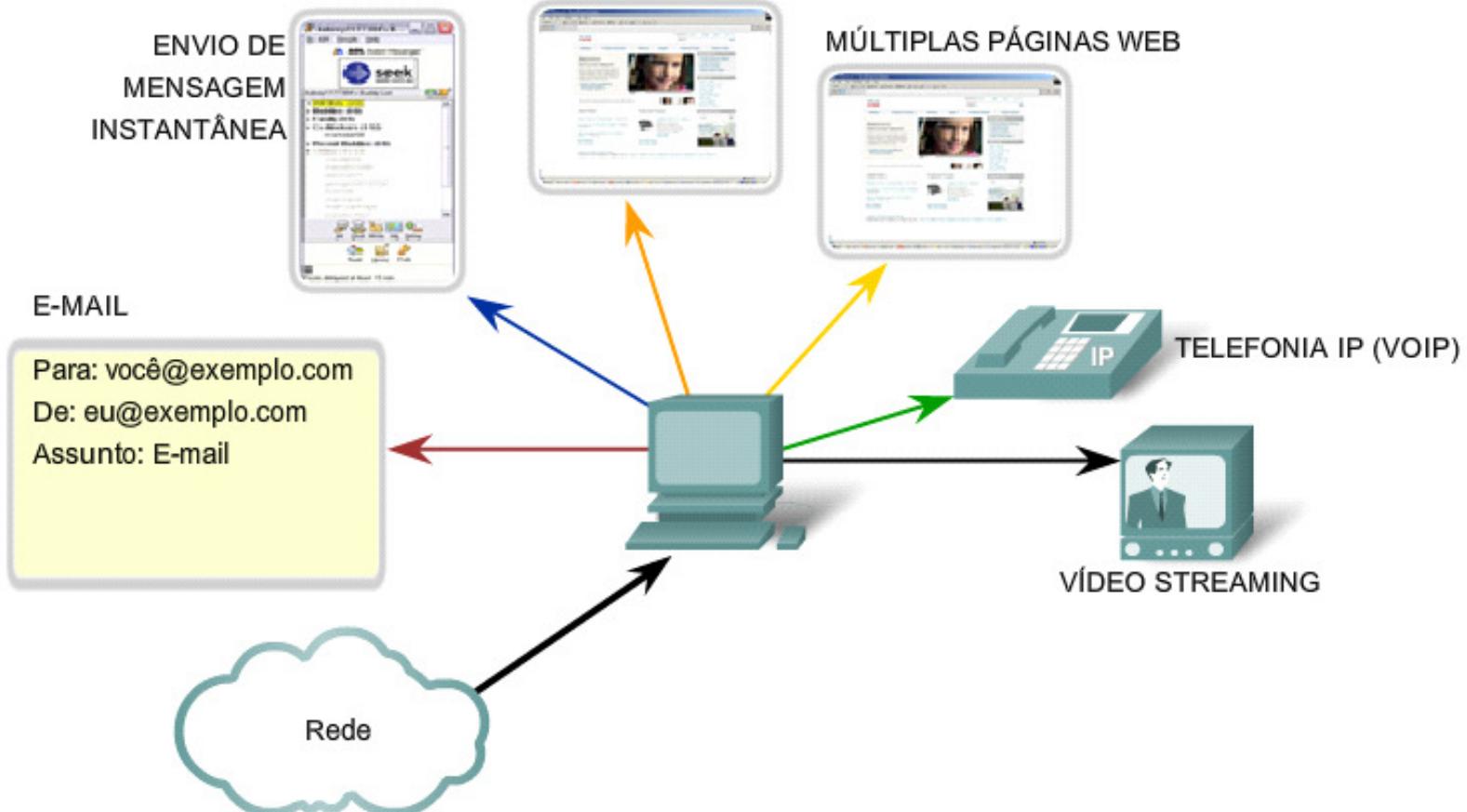
- Qualquer host pode ter múltiplas aplicações que se comunicam através da rede;
- Cada uma destas aplicações irá se comunicar com uma ou mais aplicações em hosts remotos;
- É responsabilidade da camada de Transporte **manter múltiplos fluxos** de comunicação entre estas aplicações.

● Identificação das Aplicações

- Para passar os fluxos de dados para as aplicações apropriadas, a camada de Transporte deve identificar a aplicação de destino.
- O TCP/IP utiliza um número de porta para identificar as aplicações.

Funções da Camada de Transporte

Rastreando as Conversas



A camada de Transporte segmenta os dados e gerencia a separação de dados para diferentes aplicações.
Múltiplas aplicações sendo executadas em um dispositivo recebem os dados corretamente.

Funções da Camada de Transporte

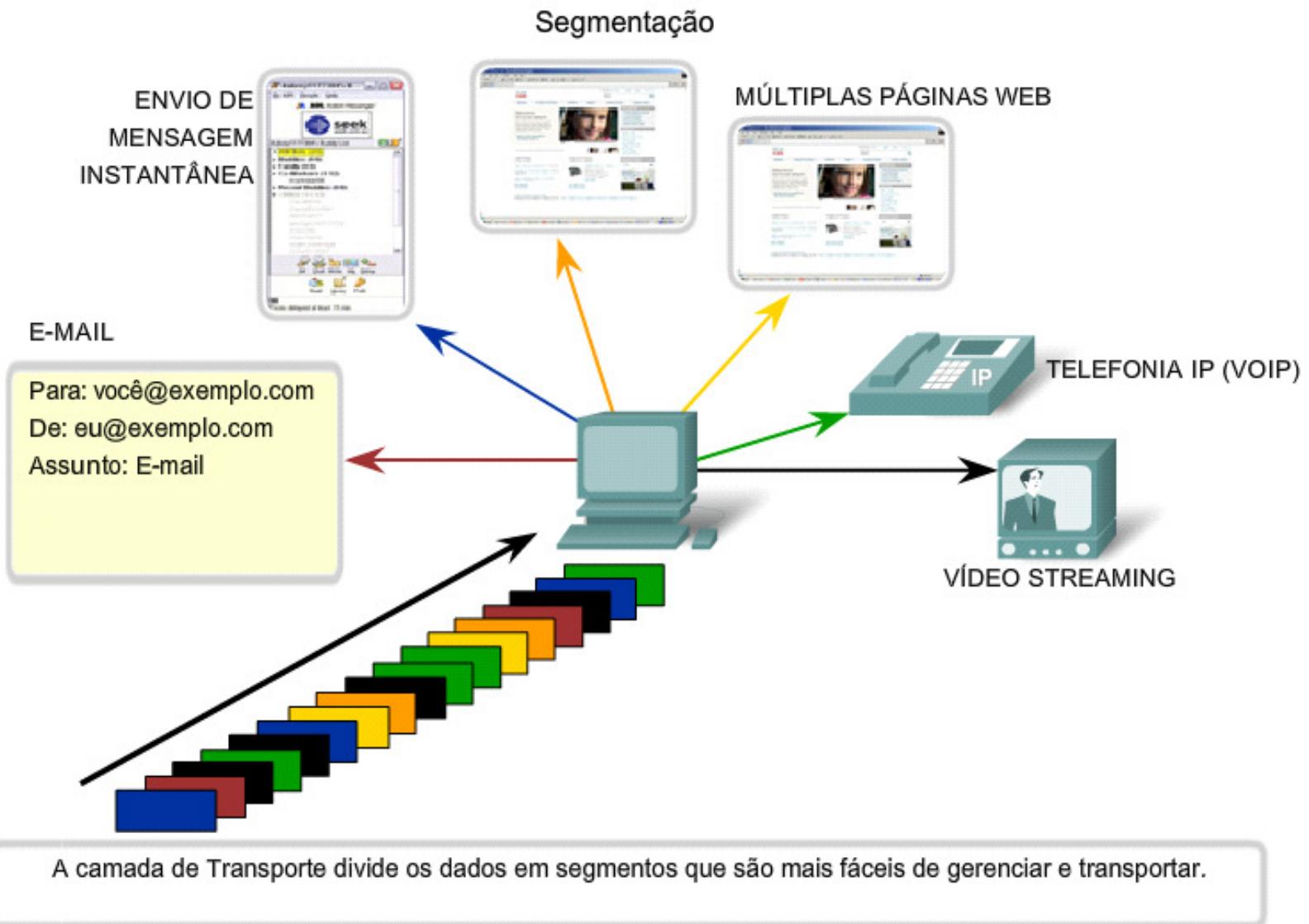
● Segmentação de Dados

- Os dados das aplicações devem ser preparados para serem enviados através do meio em **segmentos gerenciáveis**;
- Cada segmento de dados de aplicação requer a **adição de cabeçalhos** da camada de Transporte para indicar a qual comunicação ele está associado.

● Resegmentação de Dados

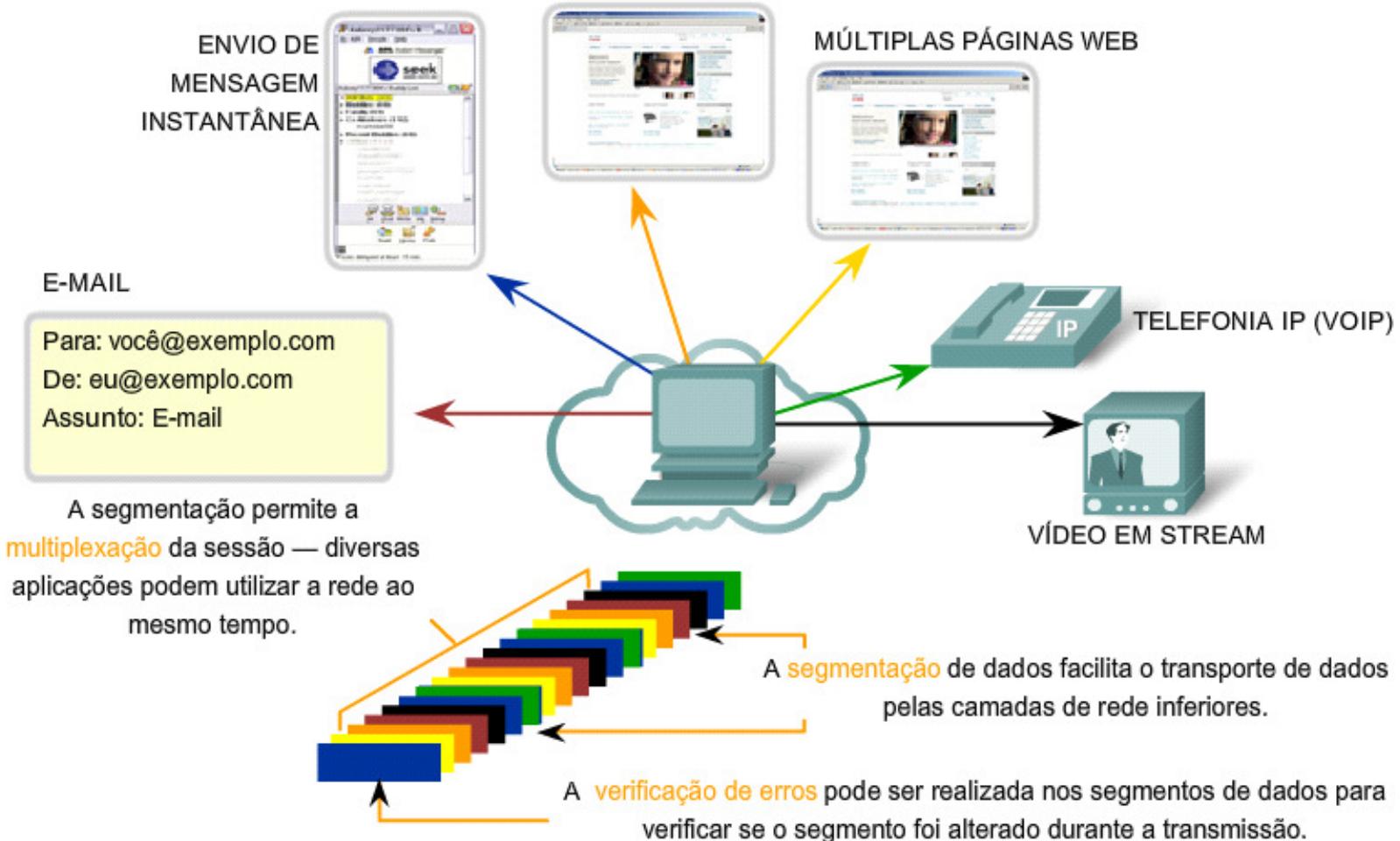
- No host de destino, cada segmento de dados pode ser **direcionado** para a aplicação apropriada;
- Estes segmentos de dados individuais também precisam ser **reconstruídos em um fluxo completo** de dados que seja útil para a camada de Aplicação.

Funções da Camada de Transporte



Controle de Conversação

Serviços da Camada de Transporte



Controle da Conversação

Serviços da Camada de Transporte



Conteúdo

1 Fundamentos

- Funções da Camada de Transporte
- Relações entre as Camadas de Transporte e Rede
- A Camada de Transporte na Internet
- Multiplexação e Demultiplexação
- Segmentação e Reagrupamento
- Endereçamento de Porta

2 Transporte sem Conexão: UDP

3 Princípios da Transferência Confiável de Dados

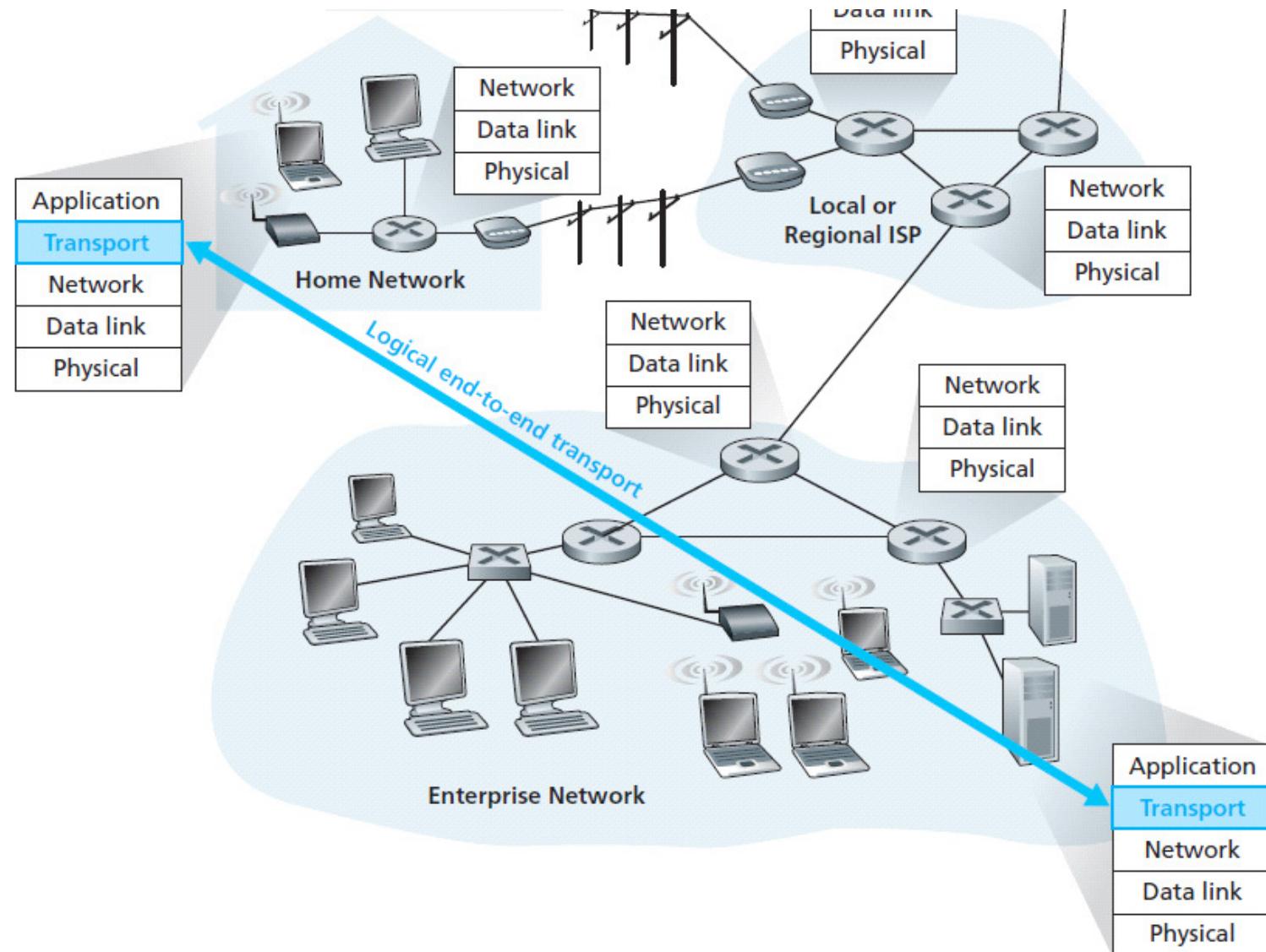
4 Transporte Orientado à Conexão: TCP

5 Controle de Congestionamento

Relações entre as Camadas de Transporte e Rede

- Um protocolo de camada de Transporte fornece **comunicação lógica entre processos** de aplicação que rodam em hosts diferentes;
- Um protocolo de camada de Rede fornece **comunicação lógica entre hosts**;
- Analogia com moradias:
 - Mensagens de aplicação → cartas em envelopes;
 - Processos → pessoas;
 - Hosts (sistemas finais) → casas;
 - Protocolo de camada de transporte → Moradores responsáveis por reunir e distribuir as cartas da casa;
 - Protocolo de camada de rede → serviço postal (incluindo os carteiros).

Relações entre as Camadas de Transporte e Rede



Conteúdo

1 Fundamentos

- Funções da Camada de Transporte
- Relações entre as Camadas de Transporte e Rede
- A Camada de Transporte na Internet
- Multiplexação e Demultiplexação
- Segmentação e Reagrupamento
- Endereçamento de Porta

2 Transporte sem Conexão: UDP

3 Princípios da Transferência Confiável de Dados

4 Transporte Orientado à Conexão: TCP

5 Controle de Congestionamento

A Camada de Transporte na Internet

- A Internet disponibiliza dois protocolos de transporte distintos para a camada de aplicação:
 - **UDP (User Datagram Protocol - Protocolo de Datagrama de Usuário)**
 - Provê à aplicação solicitante um serviço não-confiável, não orientado para conexão.
 - **TCP (Transmission Control Protocol - Protocolo de Controle de Transmissão)**
 - Provê à aplicação solicitante um serviço confiável, orientado para conexão.

Conteúdo

1 Fundamentos

- Funções da Camada de Transporte
- Relações entre as Camadas de Transporte e Rede
- A Camada de Transporte na Internet
- Multiplexação e Demultiplexação
- Segmentação e Reagrupamento
- Endereçamento de Porta

2 Transporte sem Conexão: UDP

3 Princípios da Transferência Confiável de Dados

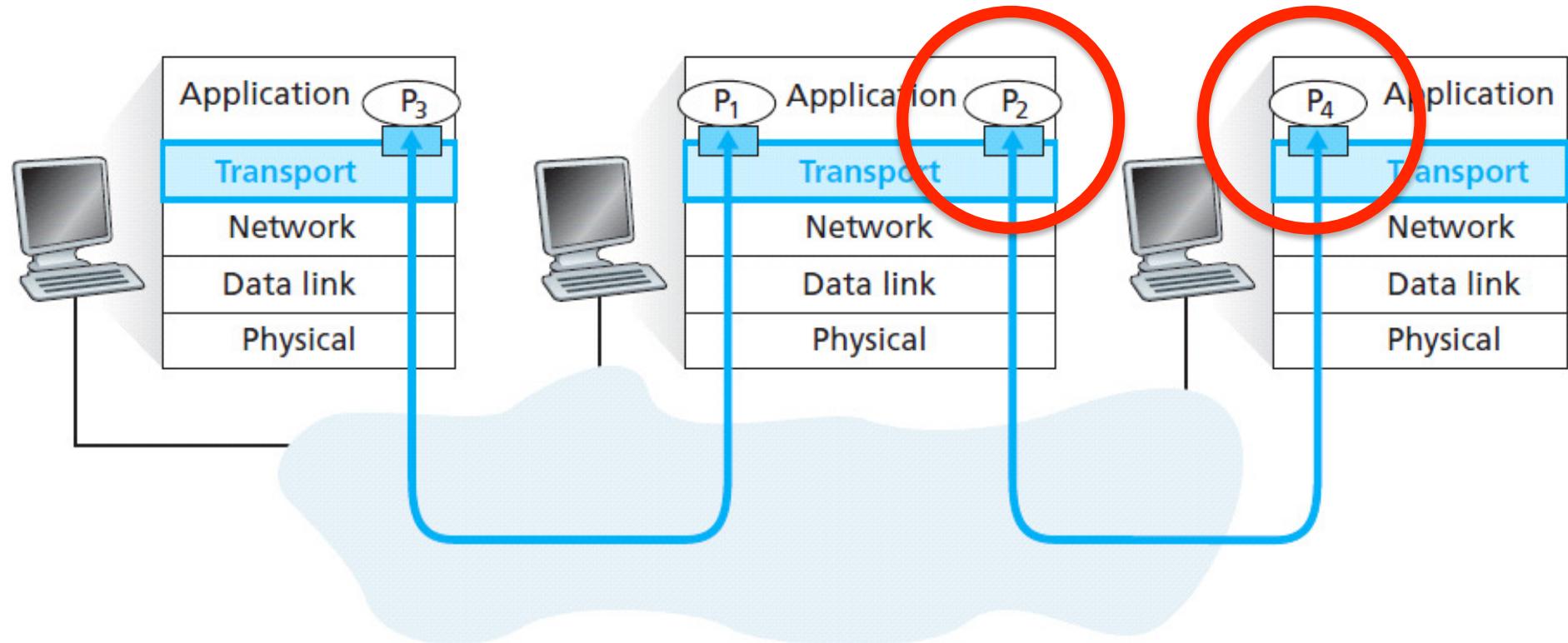
4 Transporte Orientado à Conexão: TCP

5 Controle de Congestionamento

Multiplexação e Demultiplexação

- A ampliação da entrega host-a-host para entrega processo-a-processo é denominada **multiplexação/demultiplexação de camada de transporte**;
- O trabalho de reunir porções de dados provenientes de diferentes portas, encapsular cada porção com informações de cabeçalho, e passar os segmentos para a camada de rede é denominada multiplexação;
- A tarefa de entregar os dados contidos em um segmento da camada de transporte à porta correta é denominada demultiplexação.

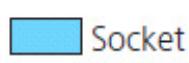
Multiplexação e Demultiplexação



Key:



Process



Socket

Conteúdo

1 Fundamentos

- Funções da Camada de Transporte
- Relações entre as Camadas de Transporte e Rede
- A Camada de Transporte na Internet
- Multiplexação e Demultiplexação
- Segmentação e Reagrupamento
- Endereçamento de Porta

2 Transporte sem Conexão: UDP

3 Princípios da Transferência Confiável de Dados

4 Transporte Orientado à Conexão: TCP

5 Controle de Congestionamento

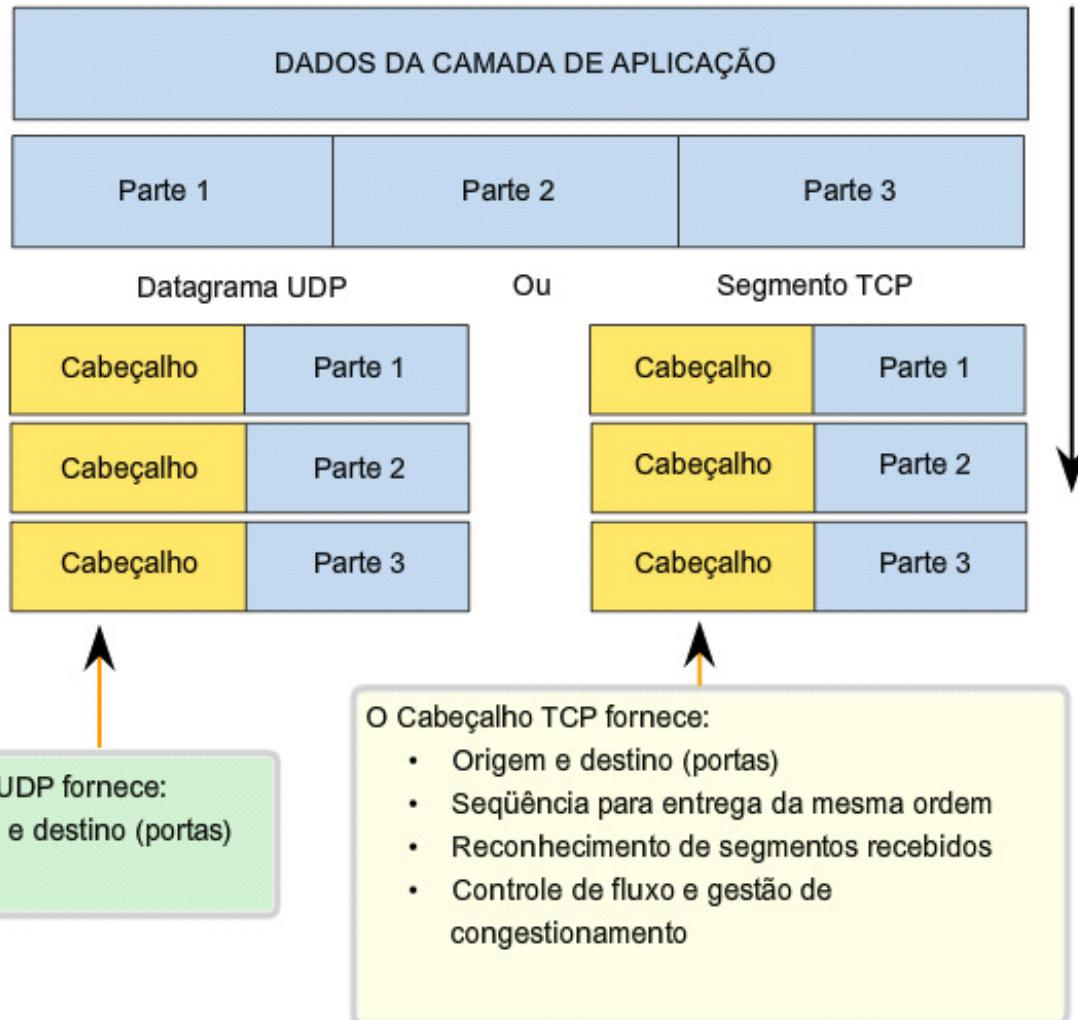
Segmentação e Reagrupamento

- Dividir os dados da aplicação em segmentos assegura que os dados sejam transmitidos dentro dos limites do meio e que os dados de diferentes aplicações possam ser multiplexadas no meio.

Segmentação e Reagrupamento

A Camada de Transporte divide os dados em partes e adiciona um cabeçalho para entrega pela rede.

Funções da Camada de Transporte



Conteúdo

1 Fundamentos

- Funções da Camada de Transporte
- Relações entre as Camadas de Transporte e Rede
- A Camada de Transporte na Internet
- Multiplexação e Demultiplexação
- Segmentação e Reagrupamento
- Endereçamento de Porta

2 Transporte sem Conexão: UDP

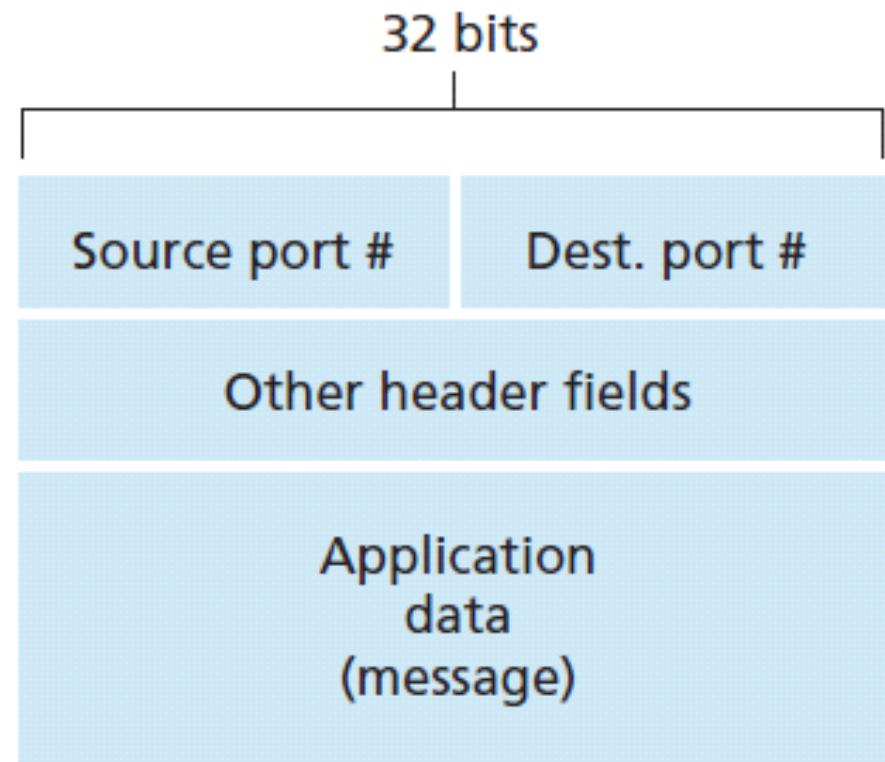
3 Princípios da Transferência Confiável de Dados

4 Transporte Orientado à Conexão: TCP

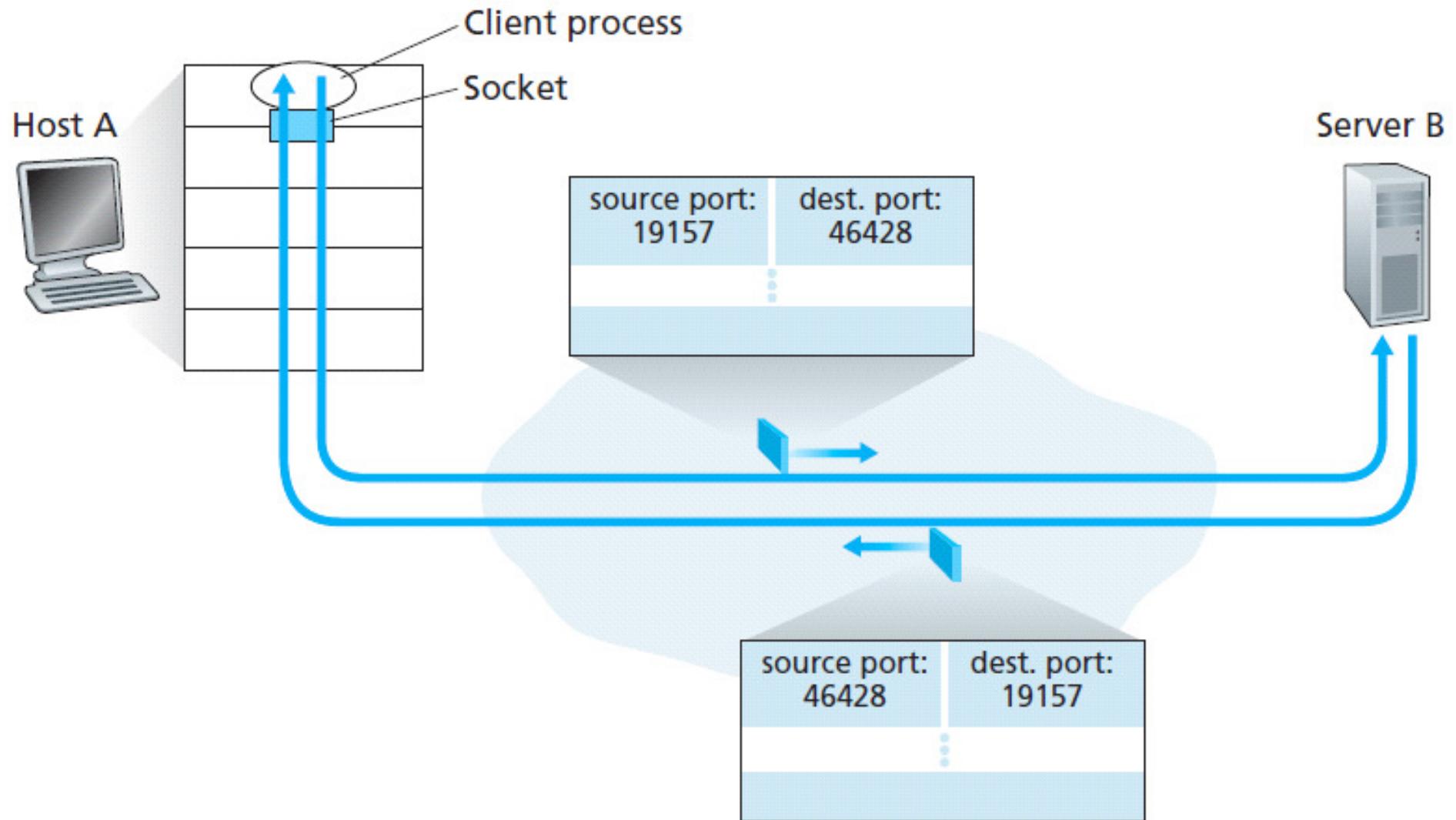
5 Controle de Congestionamento

Endereçamento de Porta

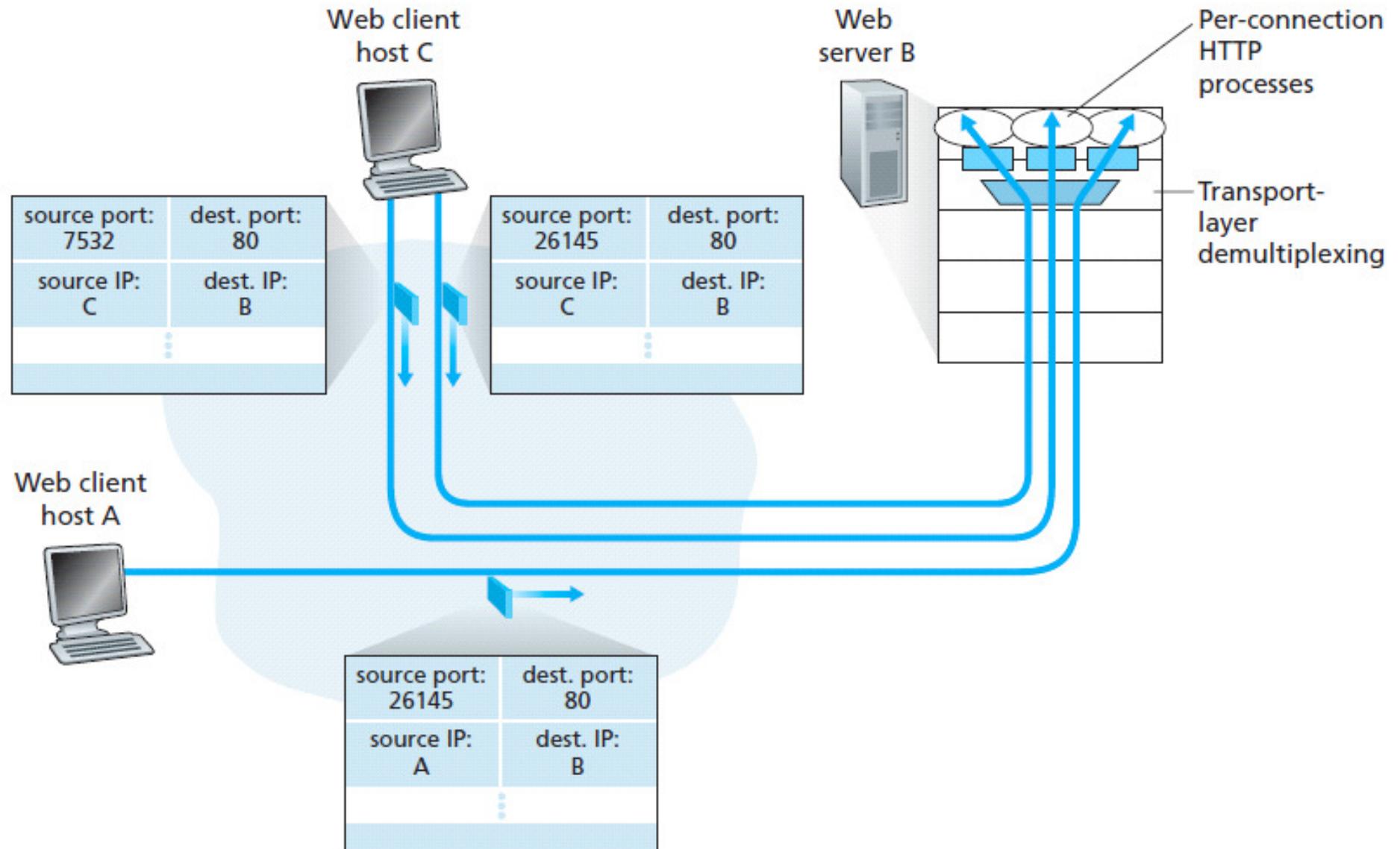
- As portas têm **identificadores exclusivos**;
- Cada segmento tem campos especiais que indicam a porta para a qual o segmento deve ser entregue:
 - Campo de número de porta da fonte;
 - Campo de número de porta do destino.
- Cada número de porta é um número de 16 bits (0 a 65535);
- Os números de porta entre 0 e 1023 são denominados **números de porta bem conhecidos**.



Endereçamento de Porta



Endereçamento de Porta



Endereçamento de Porta

Cabeçalhos TCP e UDP

Segmento TCP

Bit (0)	Bit (15) Bit (16)	Bit (31)
Porta de Origem (16)		Porta de Destino (16)
Número de Seqüência (32)		
Número de Reconhecimento (32)		
Comprimento do Cabeçalho (4) Reservado (6) Bits de Código (6)	Janela (16)	
Checksum (16)	Urgente (16)	
Opções (0 ou 32, se houver)		
DADOS DA CAMADA DE APLICATIVOS (Tamanho varia)		

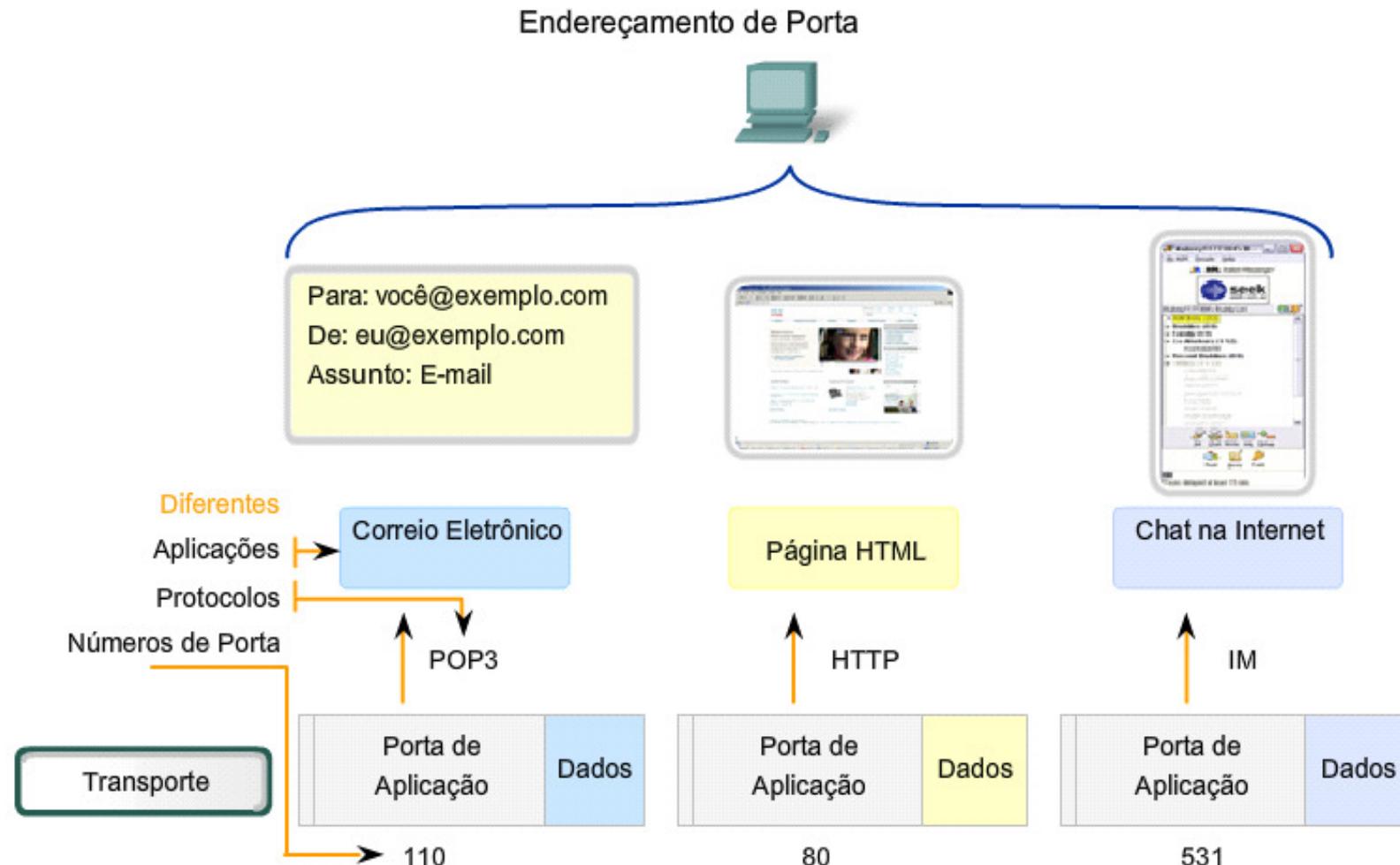
↑
20 Bytes
↓

Datagrama UDP

Bit (0)	Bit (15) Bit (16)	Bit (31)
Porta de Origem (16)		Porta de Destino (16)
Comprimento (16)		Checksum (16)
DADOS DA CAMADA DE APLICATIVOS (Tamanho varia)		

↑
8 Bytes
↓

Endereçamento de Porta



Dados para diferentes aplicações são direcionados à aplicação correta porque cada aplicação tem um número de porta único.

Endereçamento de Porta

Números das Portas

Faixa de Números de Portas	Grupo de Portas
0 a 1023	Portas conhecidas (Contato)
1024 a 49151	Portas Registradas
49152 a 65535	Portas Privadas e/ou Dinâmicas

Portas TCP Registradas:

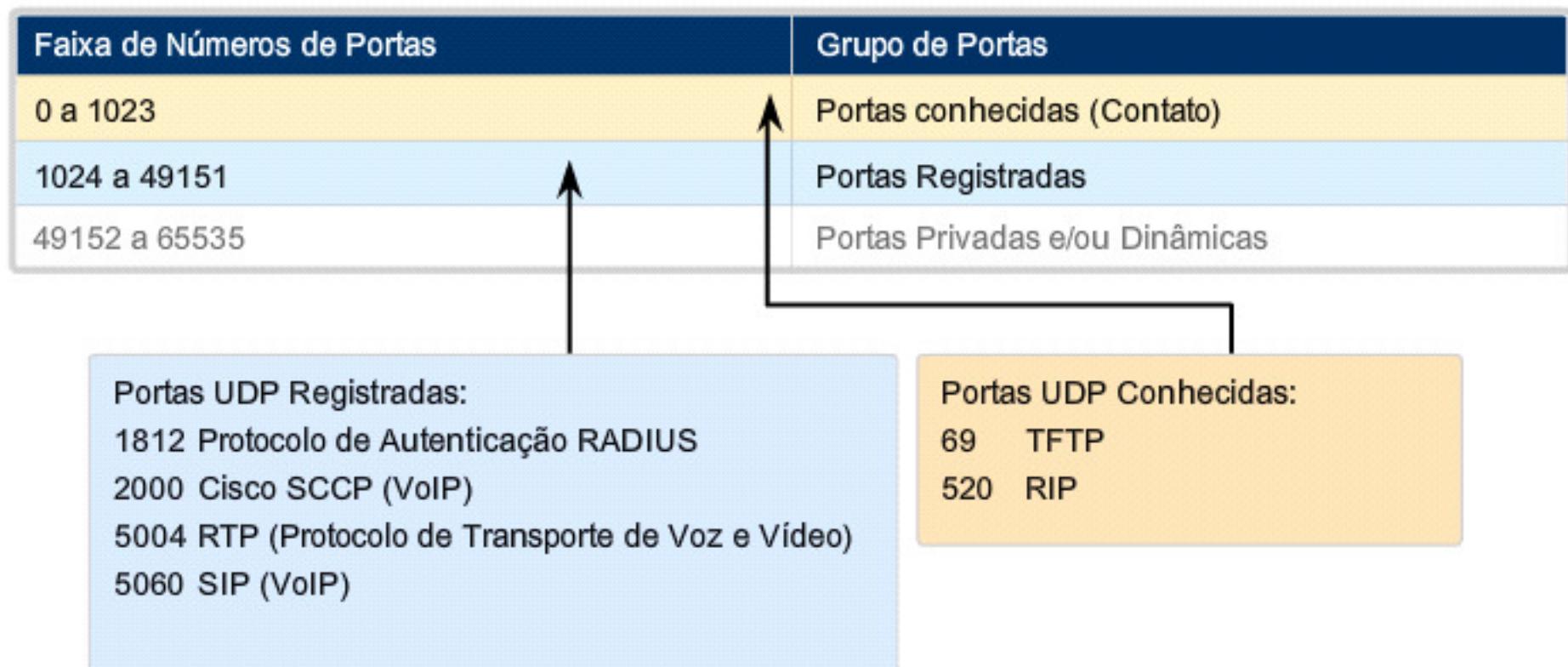
- 1863 MSN Messenger
- 8008 Alternar HTTP
- 8080 Alternar HTTP

Portas TCP Conhecidas:

- 21 FTP
- 23 Telnet
- 25 SMTP
- 80 HTTP
- 110 POP3
- 194 Internet Relay Chat (IRC)
- 443 HTTP Seguro (HTTPS)

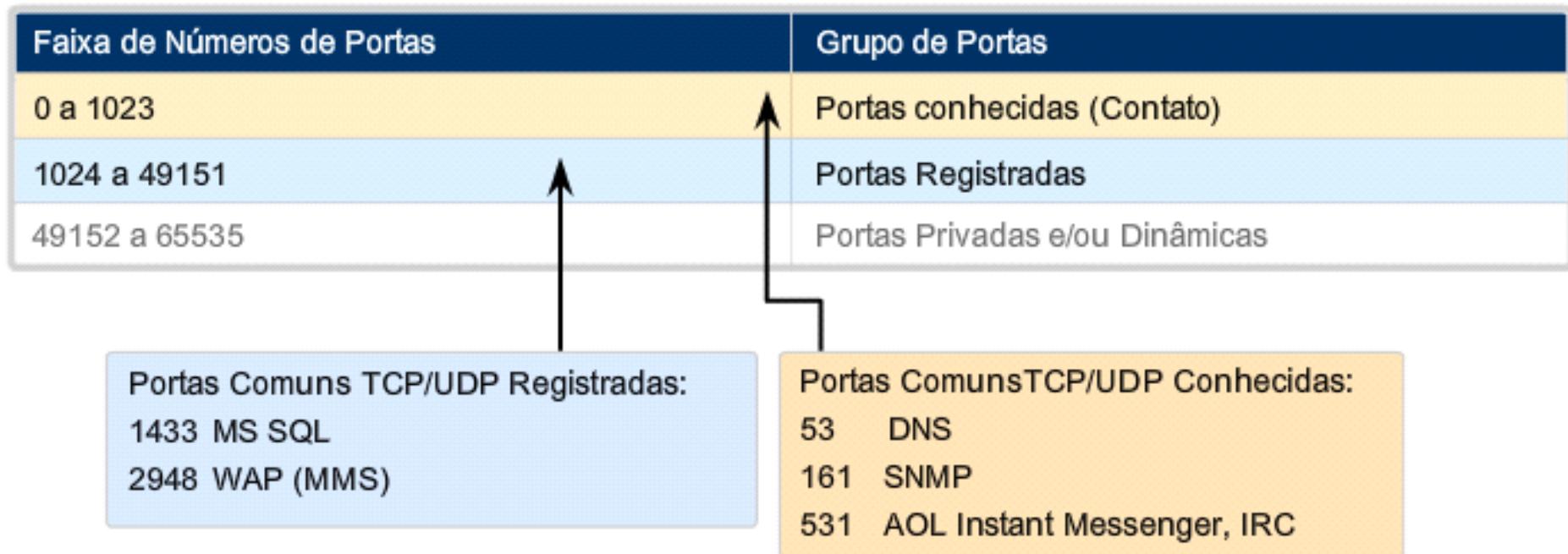
Endereçamento de Porta

Números das Portas



Endereçamento de Porta

Números das Portas

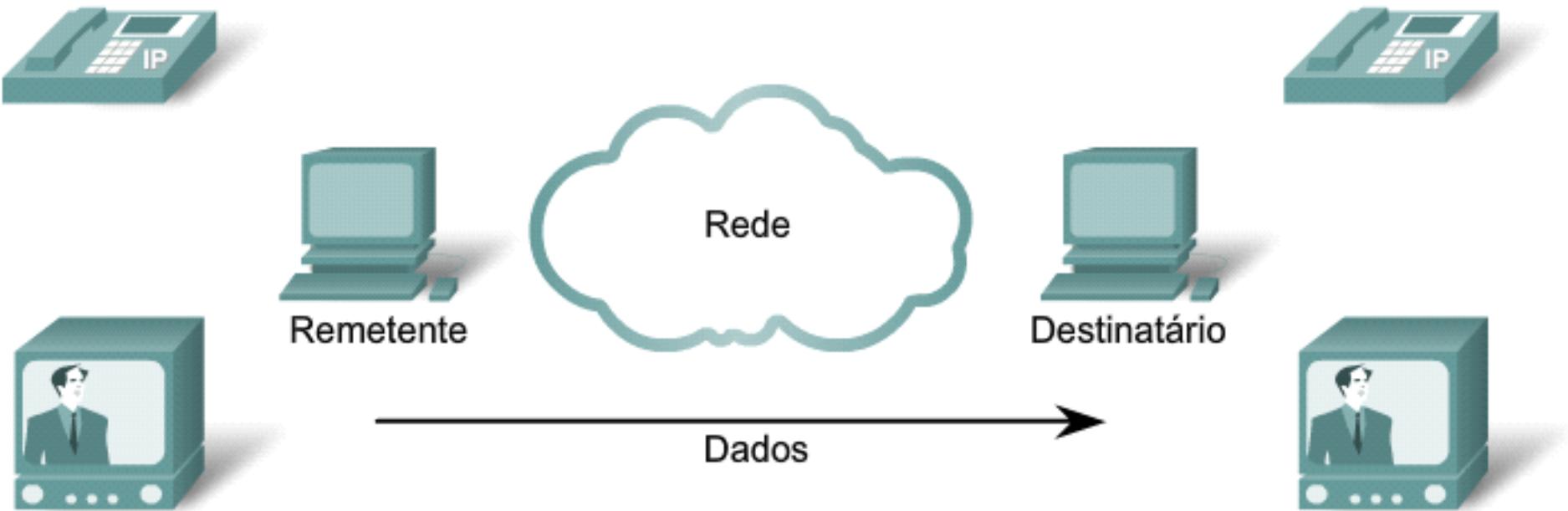


Fundamentos

- O UDP é um protocolo simples que fornece as funções básicas da camada de Transporte;
- Ele possui overhead muito mais baixo do que o TCP, já que não é orientado à conexão e não fornece mecanismos de retransmissão, sequenciamento e controle de fluxo sofisticados;
- Os principais protocolos da camada de Aplicação que usam UDP incluem:
 - Domain Name System (DNS);
 - Simple Network Management Protocol (SNMP);
 - Protocolo de Configuração Dinâmica de Host (DHCP);
 - Routing Information Protocol (RIP);
 - Trivial File Transfer Protocol (TFTP);
 - Jogos On-line.

UDP: Baixo Overhead versus Confiabilidade

Transporte de Dados Auxiliares UDP



UDP não estabelece uma conexão
antes de enviar dados.

Fundamentos

O TCP não é sempre preferível ao UDP, já que fornece serviço confiável de transferência de dados e o UDP não?

Fundamentos

O TCP não é sempre preferível ao UDP, já que fornece serviço confiável de transferência de dados e o UDP não?

- A resposta é 'não', pois muitas aplicações se adaptam melhor ao UDP pelas seguintes razões:
- **Melhor controle no nível de aplicação sobre quais dados são enviados e quando**
 - Aplicações de tempo real requerem uma taxa mínima de envio, portanto elas não querem atrasar excessivamente a transmissão de segmentos e podem tolerar uma certa perda de dados;
 - As aplicações podem usar UDP e implementar, como parte da aplicação, qualquer funcionalidade adicional necessária além do serviço de entrega simples e básico do UDP.

Fundamentos

- **Não há estabelecimento de conexão**

- O UDP simplesmente envia mensagens sem nenhuma apresentação preliminar formal e, assim, não introduz nenhum atraso para estabelecer uma conexão.

- **Não há estados de conexão**

- O UDP não monitora nenhum parâmetro, como por exemplo os buffers de envio e recebimento, parâmetros de controle de congestionamento e parâmetros numéricos de sequência e de reconhecimento;
- Por essa razão, um servidor devotado a uma aplicação específica pode suportar um número muito maior de clientes ativos quando o UDP é usado.

- **Pequena sobrecarga de cabeçalho de pacote**

- O segmento UDP tem somente 8 bytes de sobrecarga de cabeçalho, enquanto o TCP tem 20 bytes de sobrecarga.

Fundamentos

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Fundamentos

Por que seria interessante que existisse um controle de congestionamento adaptativo para o UDP?

Fundamentos

Por que seria interessante que existisse um controle de congestionamento adaptativo para o UDP?

- O controle de congestionamento é necessário para evitar que a rede entre em um estado no qual pouquíssimo trabalho útil é realizado;
- A transmissão desenfreada de vídeo com alta taxa causaria um transbordamento de pacotes nos roteadores, de forma que poucos segmentos UDP conseguiram chegar ao destino;
- As altas taxas de perda induzidas pelos remetentes UDP sem controle fariam com que os remetentes TCP reduzissem drasticamente suas taxas.

Reagrupamento de Segmentos UDP

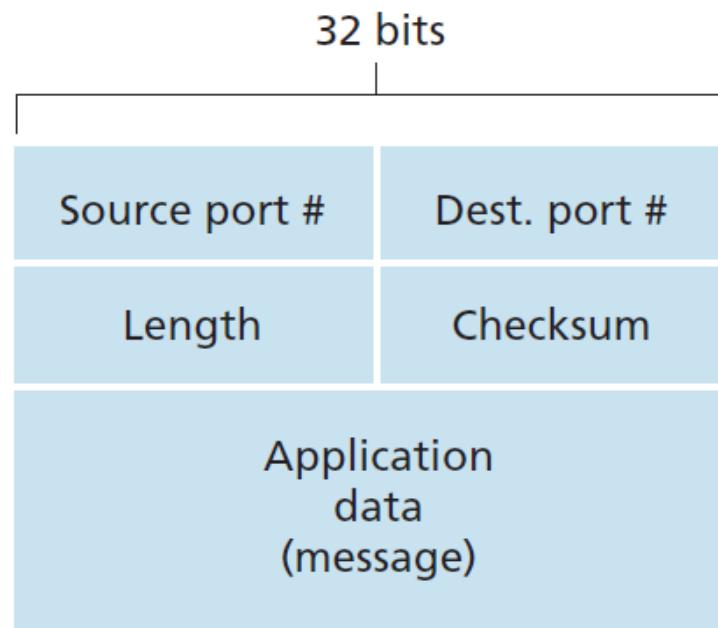
- O UDP é baseado em transação: quando uma aplicação tem dados para enviar, ele simplesmente envia os dados.
- Algumas aplicações enviarão grandes quantidades de dados que precisam ser divididos em múltiplos segmentos (segmentação);
- Quando múltiplos datagramas são enviados a um destino, eles podem tomar diferentes caminhos e chegar na ordem errada;
- O UDP não tem um modo para reordenar os datagramas na sua ordem de transmissão;
- O UDP simplesmente regrupa os dados na ordem que eles foram recebidos e os encaminha para a aplicação.

Reagrupamento de Segmentos UDP

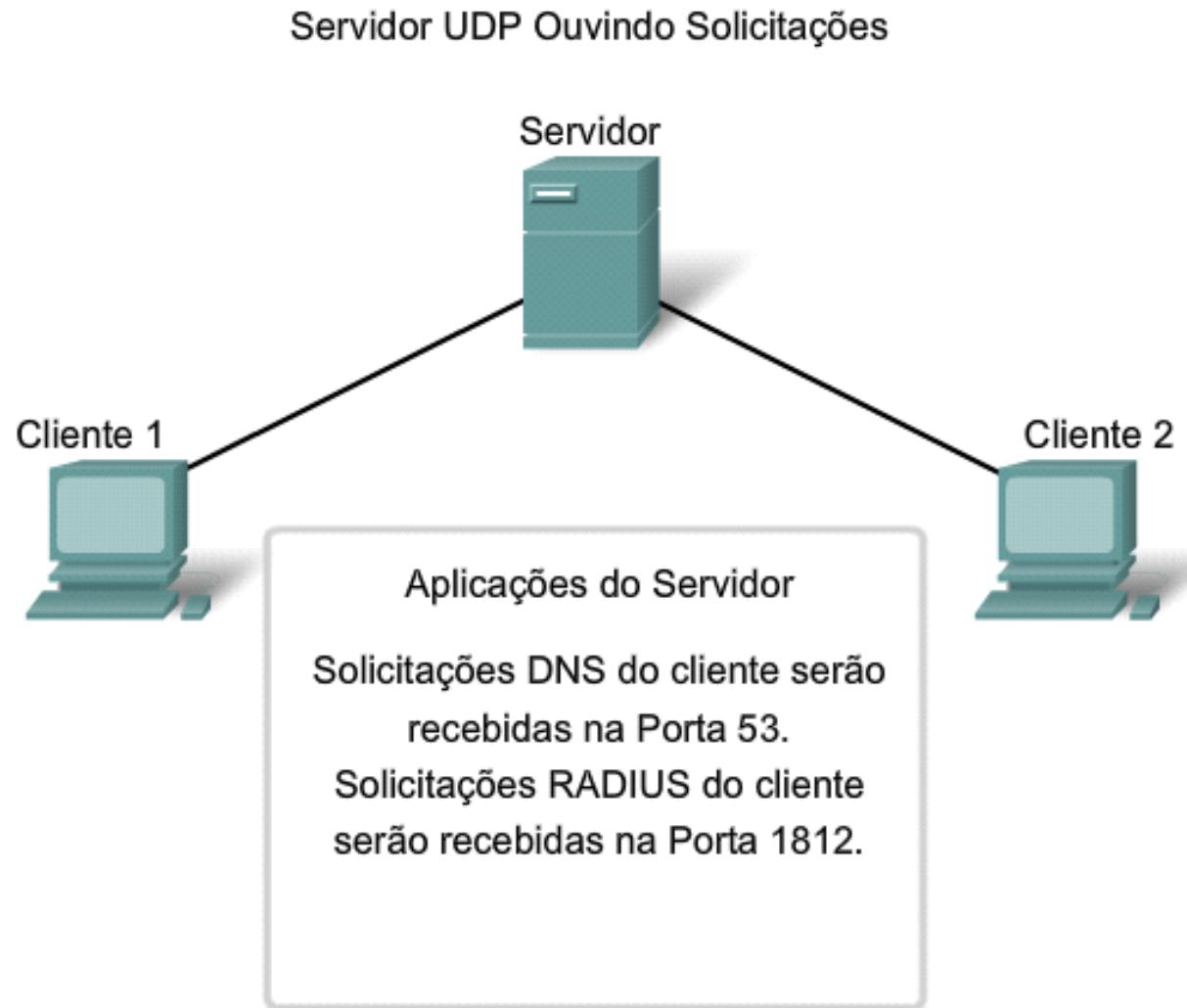


Estrutura do Segmento UDP

- A soma de verificação UDP serve para detectar erros;
- Ela é usada para determinar se bits dentro do segmento UDP foram alterados:
 - Por ruído nos enlaces;
 - Enquanto armazenados em um roteador.



Processos de Servidores e de Clientes UDP



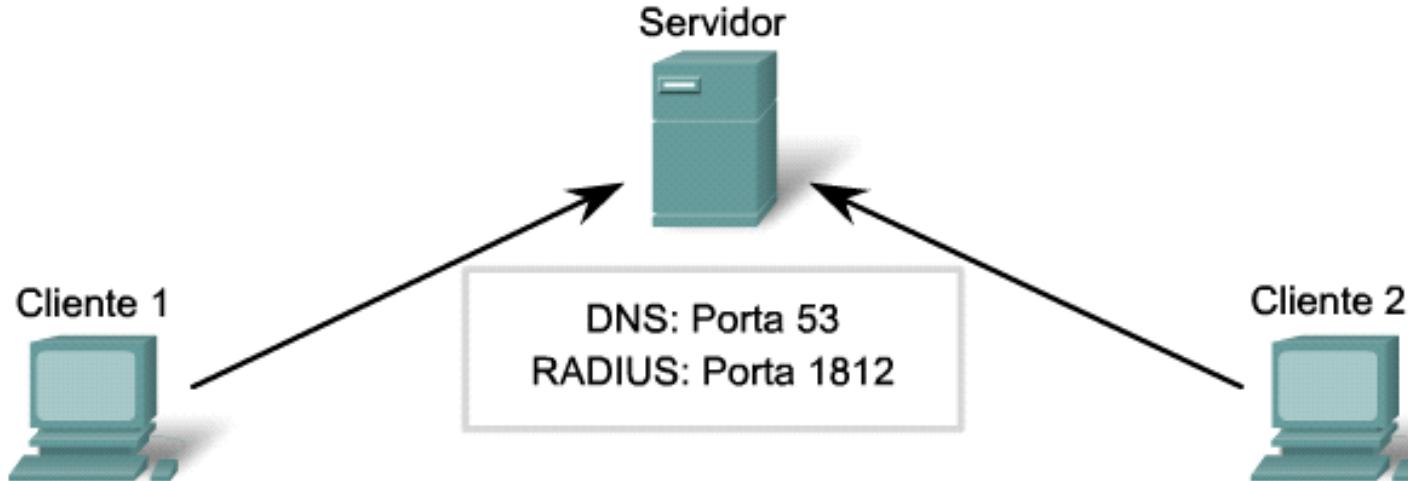
Solicitações de cliente para servidores têm números de porta conhecidos como a porta de destino.

Processos de Servidores e de Clientes UDP

- O processo cliente UDP seleciona aleatoriamente um número de porta a partir de uma faixa dinâmica de números de porta e o usa como a porta de origem para a conversação;
- A porta de destino será geralmente o número de porta Conhecida ou Registrada designado ao processo do servidor;
- Números de porta de origem randomizados também ajudam na segurança;
- Uma vez que o cliente escolheu as portas de origem e destino, o mesmo par de portas é usado no cabeçalho de todos os datagramas da transação.

Processos de Servidores e de Clientes UDP

Cientes Enviando Solicitações UDP



Solicitação de DNS do Cliente 1:

Porta de Origem 49152

Porta de Destino 53

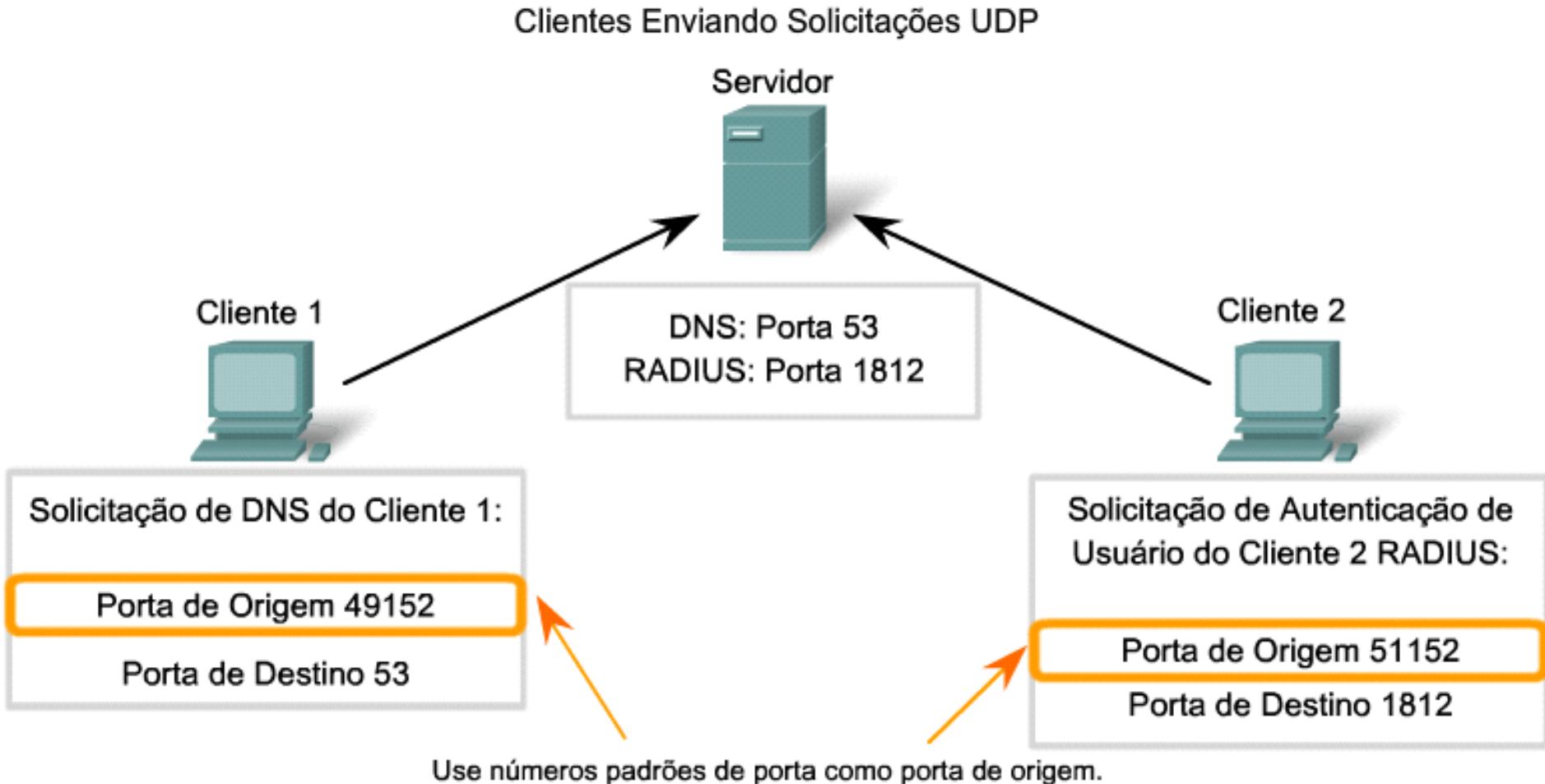
Solicitação de Autenticação de Usuário do Cliente 2 RADIUS:

Porta de Origem 51152

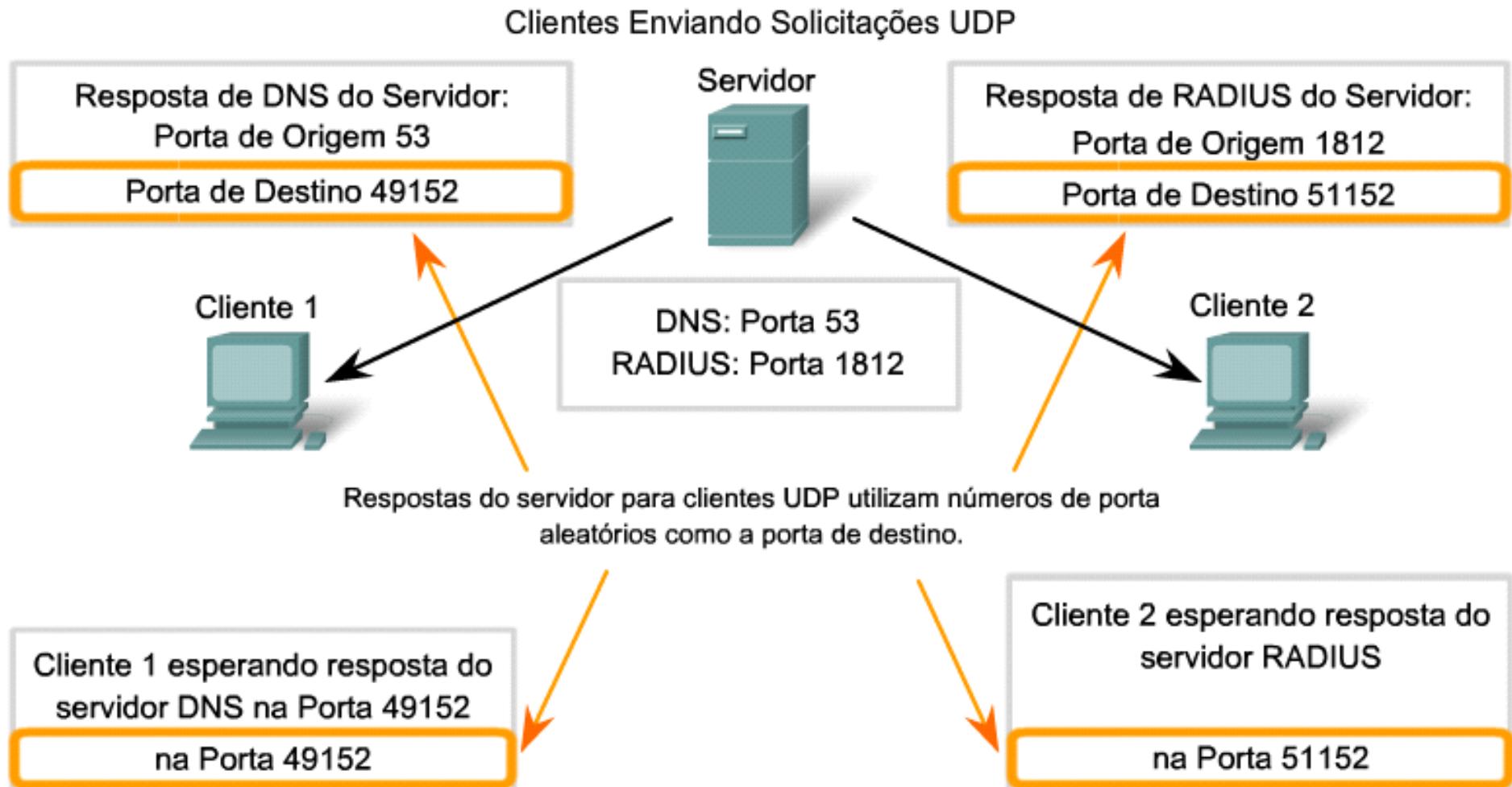
Porta de Destino 1812

Solicitações de cliente para servidor UDP utilizam números de porta conhecidos como a porta de destino.

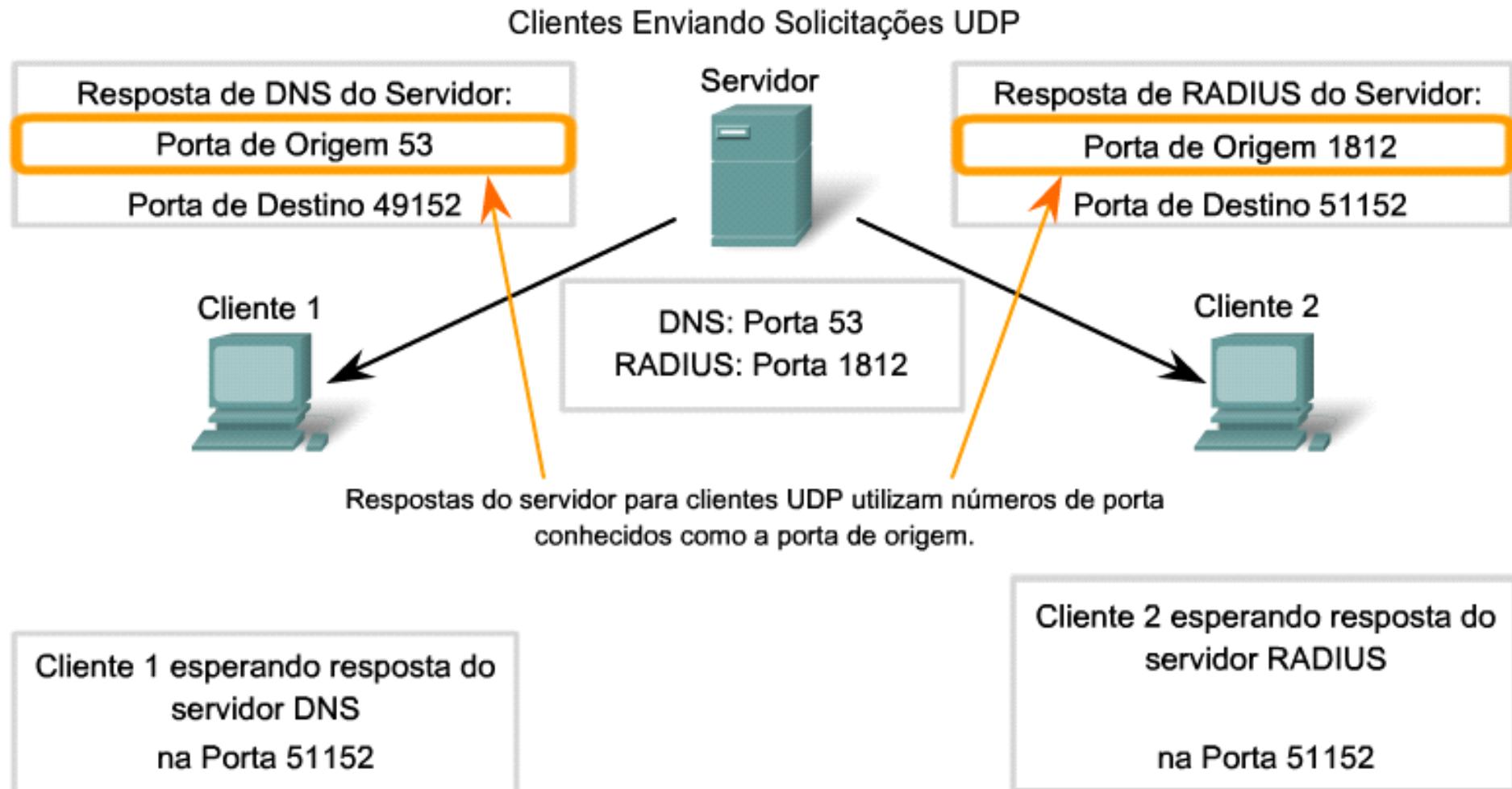
Processos de Servidores e de Clientes UDP



Processos de Servidores e de Clientes UDP



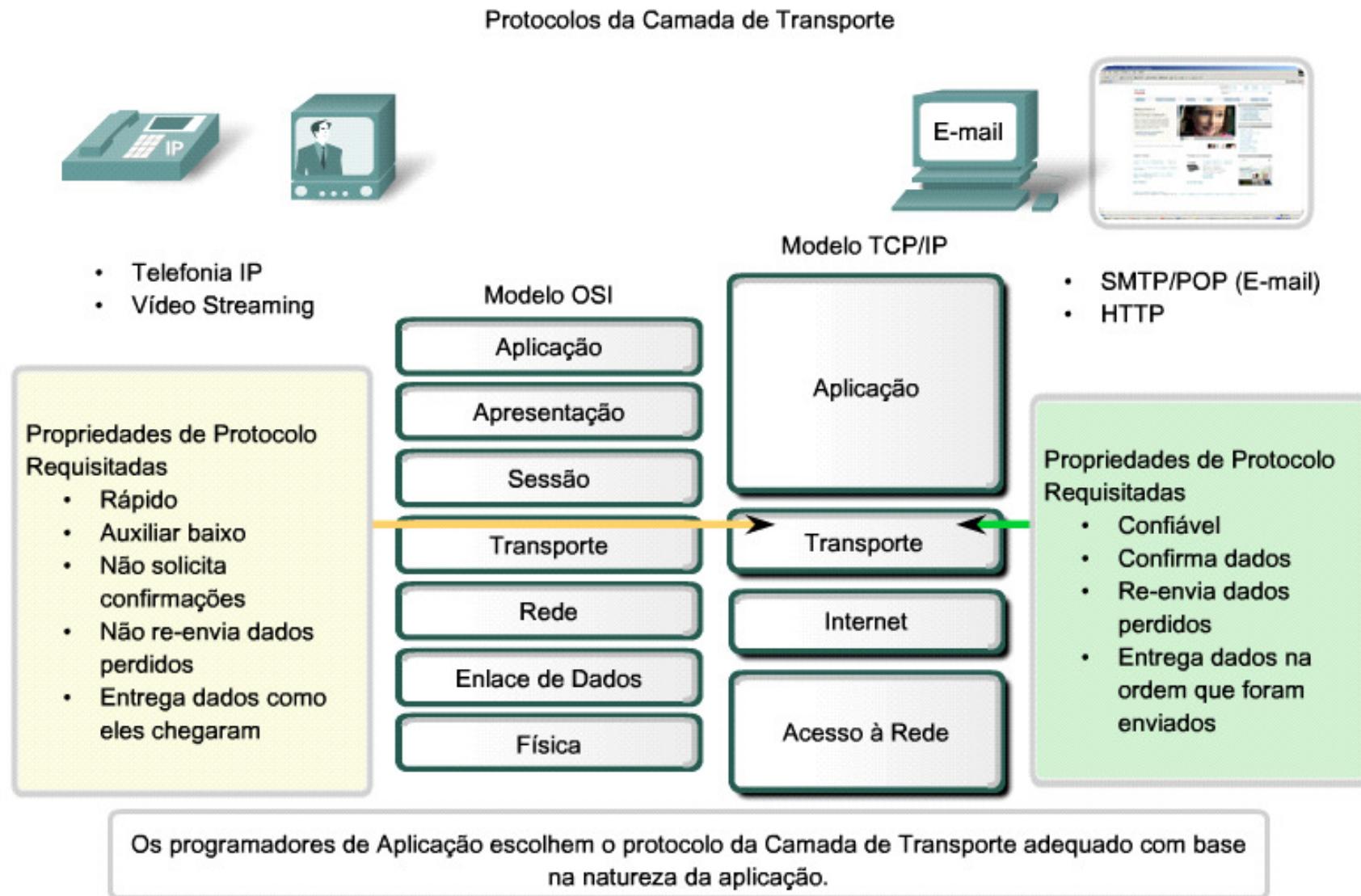
Processos de Servidores e de Clientes UDP



Transferência Confiável de Dados

- Se tivéssemos que fazer uma lista dos **dez maiores problemas** fundamentalmente importantes para o trabalho em rede, o da transferência confiável de dados seria o **candidato número um** da lista.
- Em termos de rede, confiabilidade significa assegurar que cada segmento de dado enviado pela origem chegue ao seu destino;
- Na camada de Transporte, as três operações básicas de confiabilidade são:
 - **Rastreamento** de dados transmitidos;
 - **Confirmação** de dados recebidos;
 - **Retransmissão** de quaisquer dados não confirmados.
- Para suportar estas operações de confiabilidade, mais **dados de controle** são trocados entre os hosts de envio e recepção. Esta informação de controle está contida no cabeçalho da Camada 4.

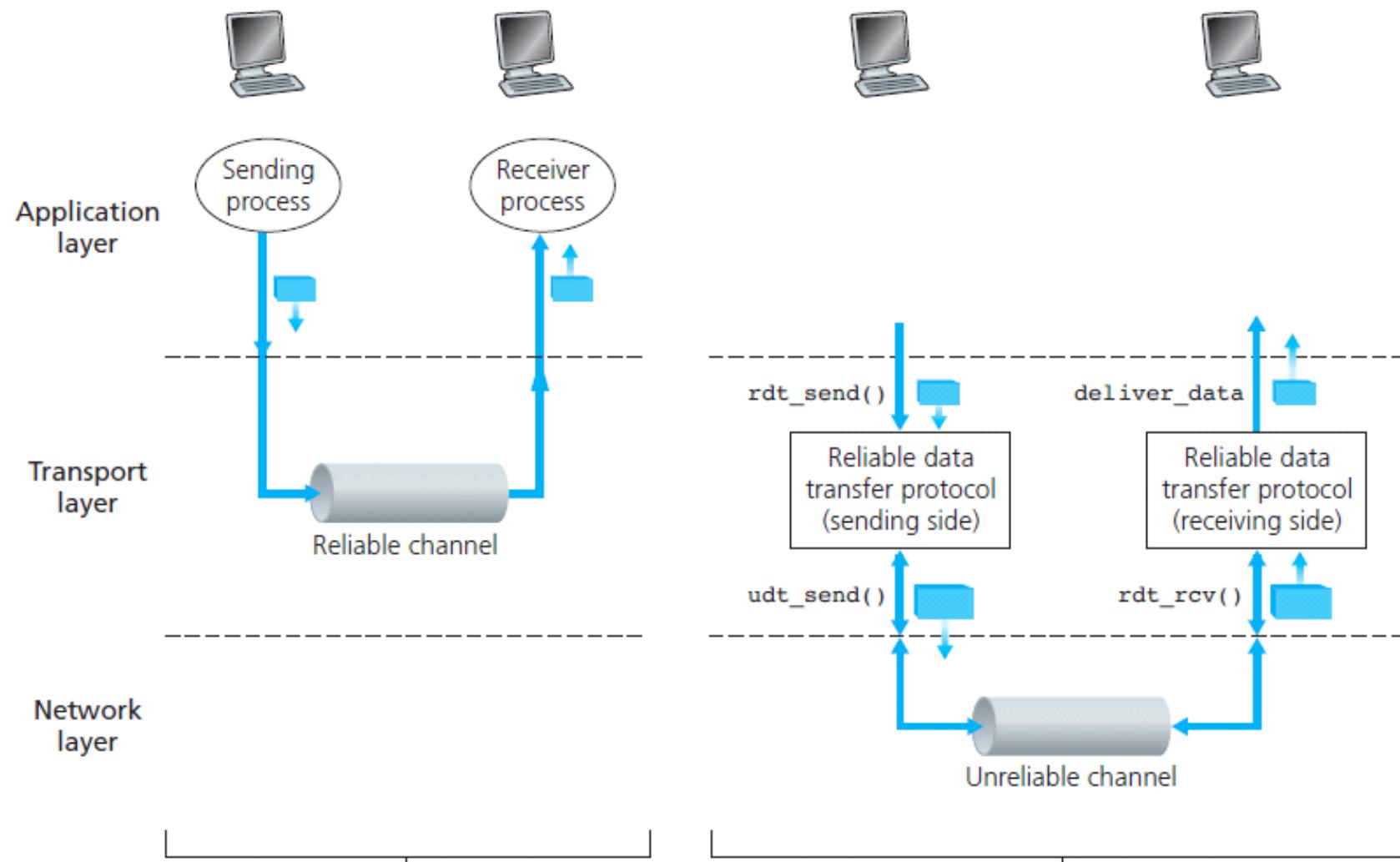
Transferência Confiável de Dados



Transferência Confiável de Dados

- O TCP oferece um **modelo de serviço** às aplicações de Internet que recorrem a ele;
- Nesse modelo, o TCP **oferece abstratamente um canal confiável** através do qual dados podem ser transferidos;
- Com um canal confiável, nenhum dos dados transferidos é **corrompido** (trocado de 0 para 1 ou vice-versa), nem **perdido**, e todos são entregues na **ordem** em que foram enviados;
- Desenvolveremos gradualmente os lados remetente e destinatário de um protocolo confiável de transferência de dados, considerando modelos progressivamente mais complexos do canal subjacente;
- Protocolo proposto: *rdt* (*Reliable Data Transfer*).

Transferência Confiável de Dados



Key:



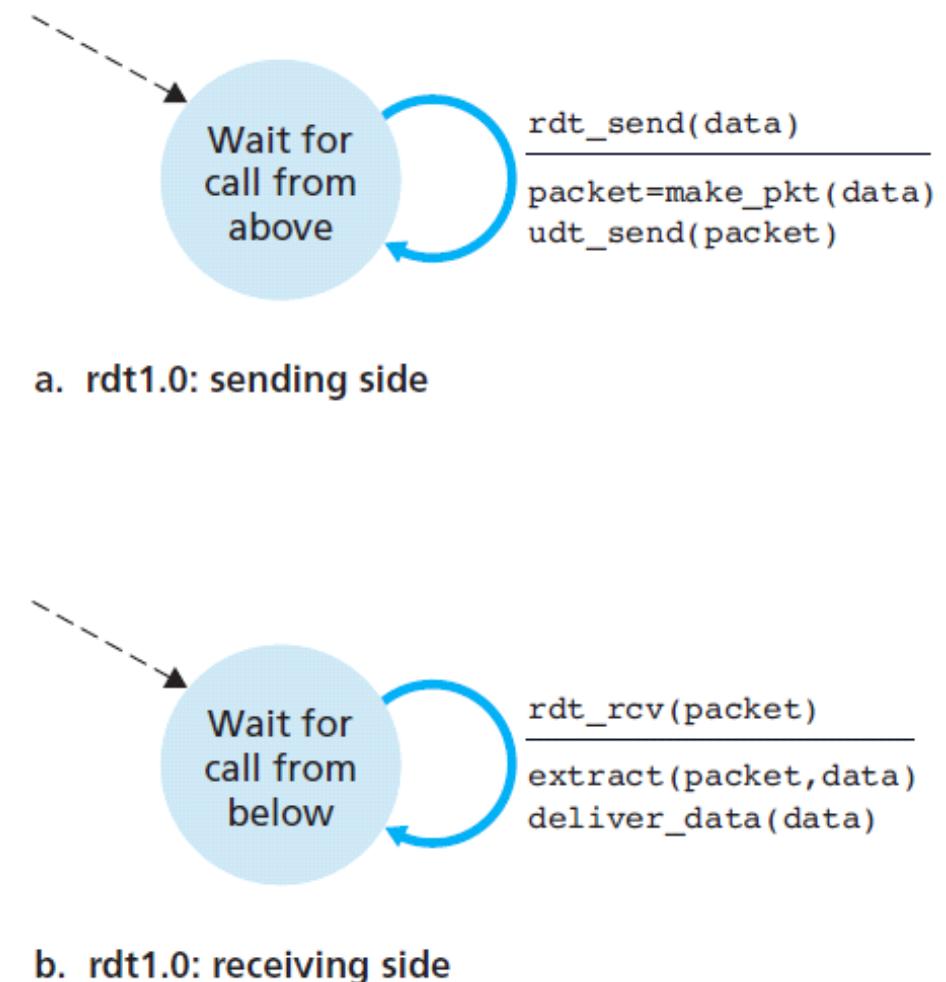
Data



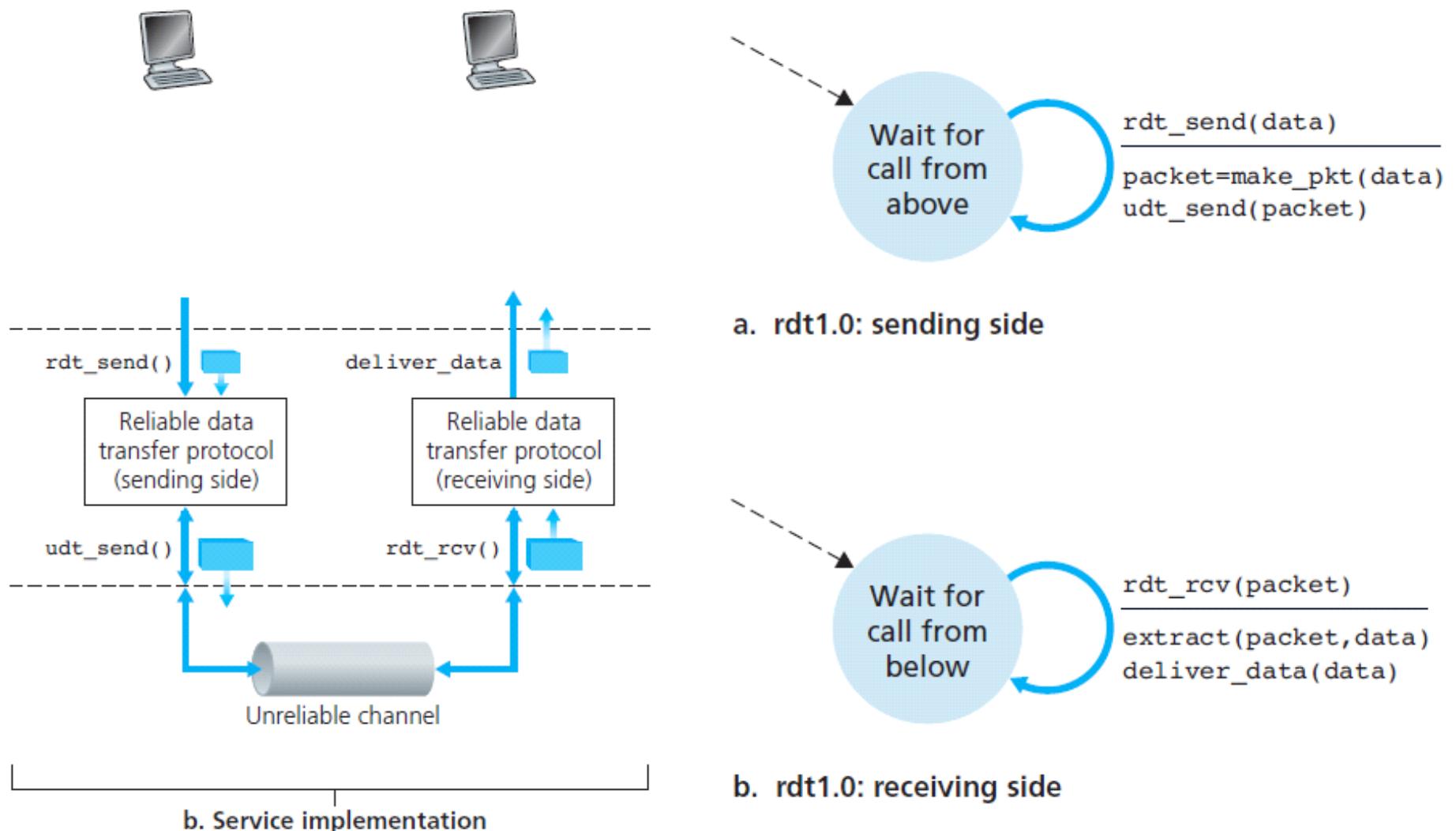
Packet

rdt1.0 - um protocolo para um canal completamente confiável

- **Máquina de Estado Finito** (Finite State Machine - FSM);
- As setas indicam a **transição** do protocolo de um estado para outro;
- O **evento** que causou a transição é mostrado acima da linha horizontal;
- As **ações** realizadas quando ocorre o evento são mostradas abaixo da linha horizontal;
- O símbolo Λ indica a falta de um evento ou uma ação;
- O estado inicial da FSM é indicado pela seta tracejada.



rdt1.0 - um protocolo para um canal completamente confiável



Transferência Confiável de Dados

Como podemos tornar o protocolo rdt1.0 mais realista?

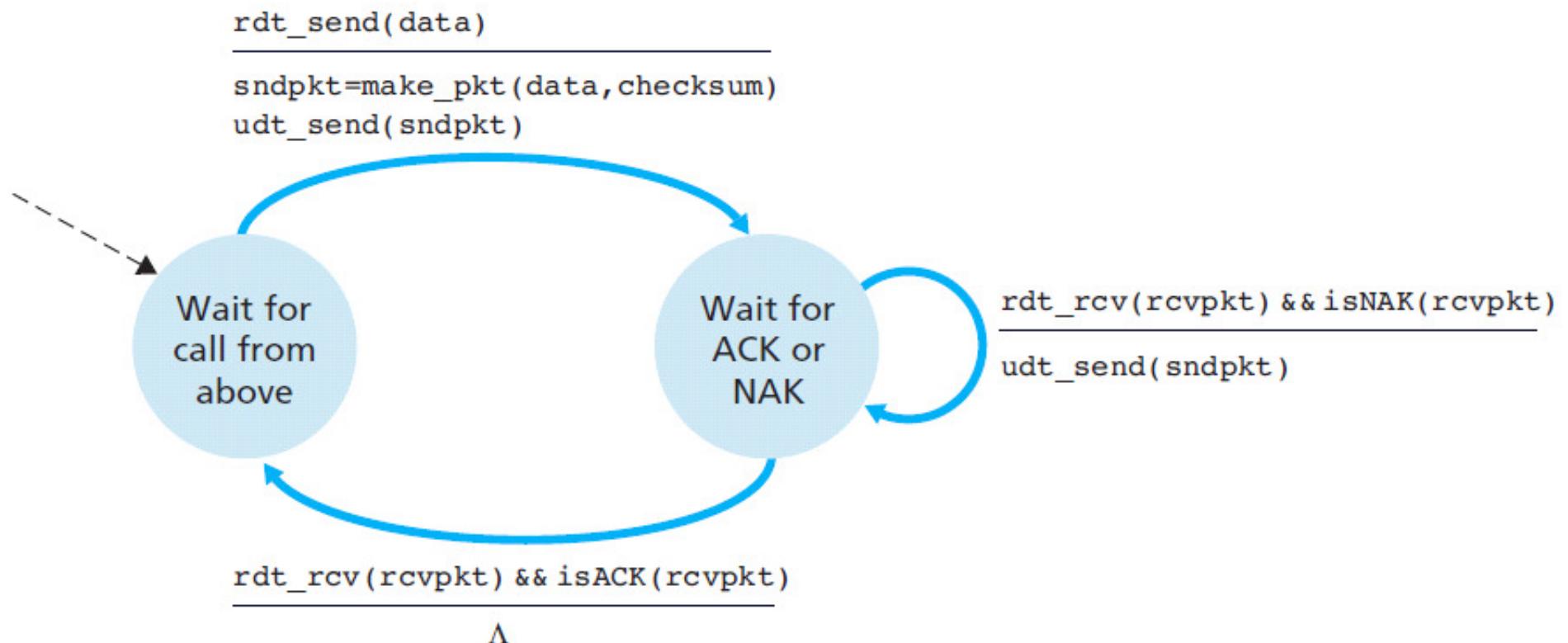
Transferência Confiável de Dados

Como podemos tornar o protocolo rdt1.0 mais realista?

Considerando que os bits de um pacote podem ser corrompidos pelo canal subjacente
- criação do protocolo rdt2.0.

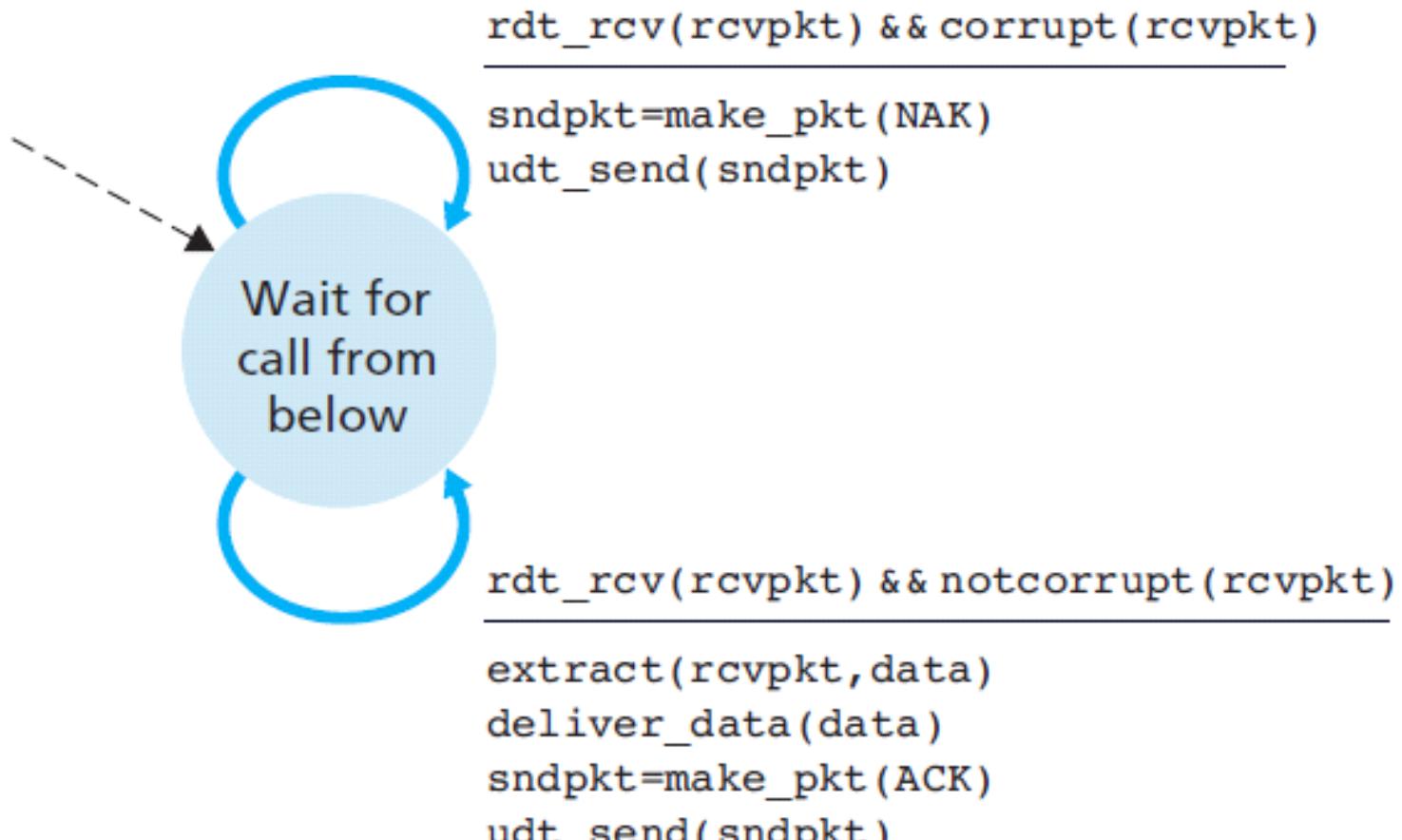
- **Protocolos ARQ** (Automatic Repeat ReQuest - solicitação automática de repetição);
- São exigidas três capacitações adicionais dos protocolos ARQ para manipular a presença de erros de bits:
 - **Detecção de erros:** **bits extras** nos cabeçalhos permitem a detecção e em alguns casos correção de erros;
 - **Realimentação do destinatário:** **reconhecimentos positivos (ACK)** e **negativos (NAK)** enviados do destinatário ao remetente;
 - **Retransmissão:** um pacote recebido com erro no destinatário é retransmitido pelo remetente.
- Protocolo **pare-e-espera**.

rdt2.0 - um protocolo para um canal com erros de bits (lado remetente)



a. rdt2.0: sending side

rdt2.0 - um protocolo para um canal com erros de bits (lado destinatário)



b. rdt2.0: receiving side

Transferência Confiável de Dados

Qual é o erro fatal do protocolo rdt2.0?

Transferência Confiável de Dados

Qual é o erro fatal do protocolo rdt2.0?

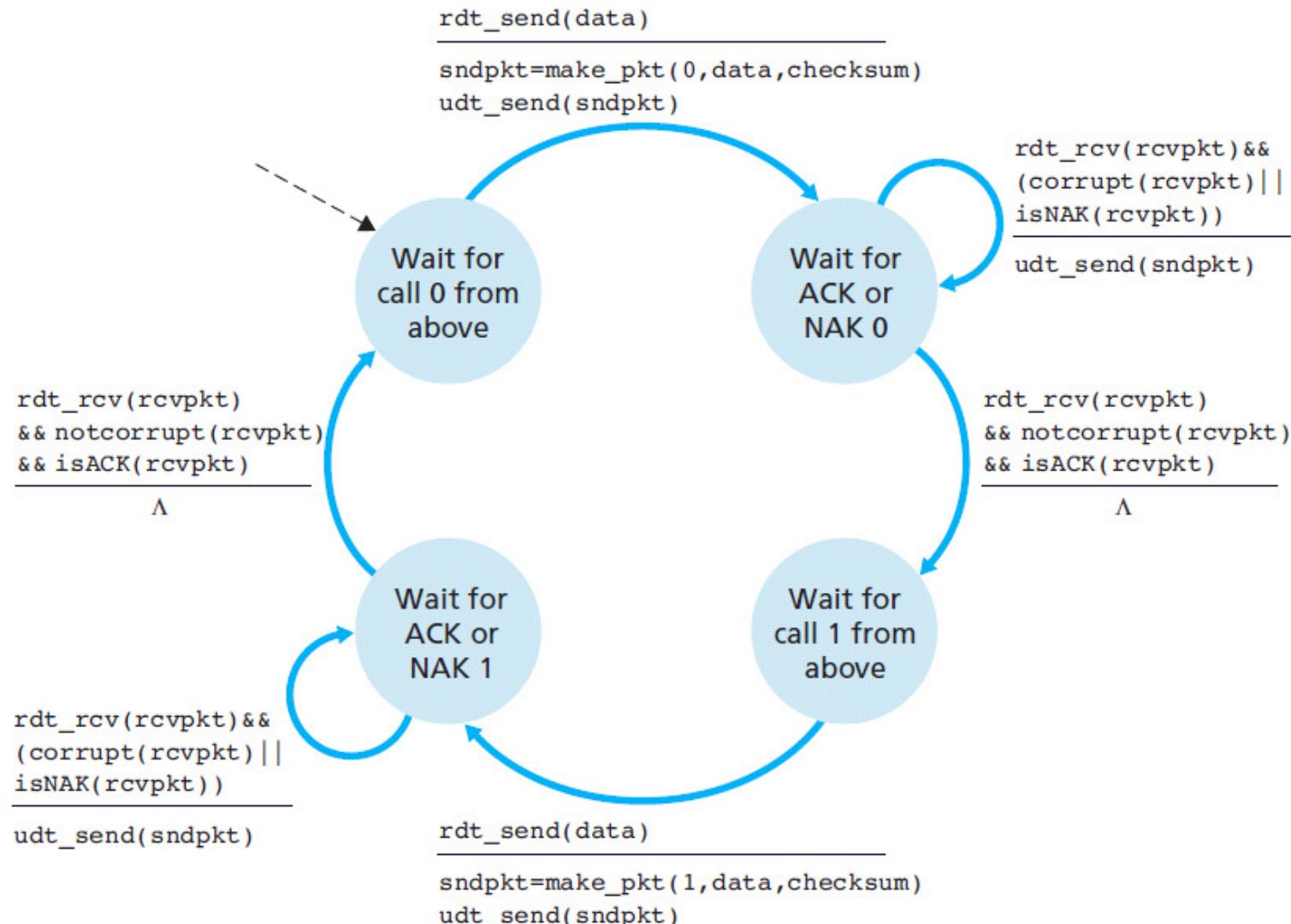
Esse protocolo não trata da possibilidade de o pacote ACK ou NAK estar corrompido.

- Podemos considerar três possibilidades para manipular ACKs e NAKs corrompidos:
 - ➊ Introduzir um novo tipo de pacote remetente-destinatário no protocolo: o que foi que você disse? E se esse novo pacote também for corrompido?
 - ➋ Adicionar um número suficiente de bits de soma de verificação para permitir a detecção e correção de erros por parte do destinatário. Resolveria o problema dos pacotes corrompidos, mas não dos perdidos;
 - ➌ O remetente simplesmente reenvia o pacote de dados corrente quando receber um ACK ou NAK truncado. Esse método introduz pacotes duplicados na rede (1^a transmissão e demais retransmissões).
- Criação do protocolo rdt2.1.

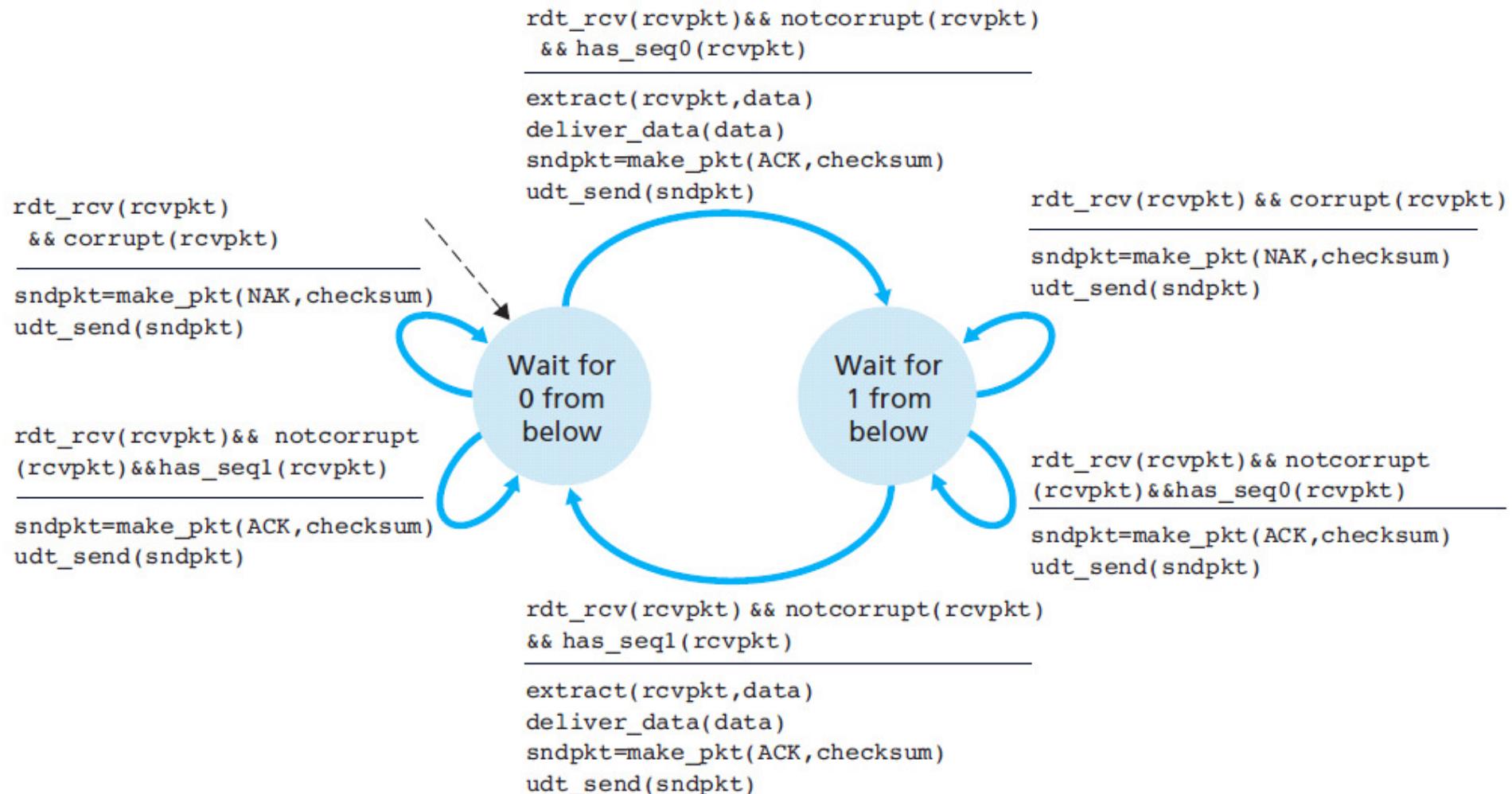
rdt2.1 - um protocolo melhorado para um canal com erros de bits (lado remetente)

- Esse protocolo faz com que o remetente numere seus pacotes de dados com um número de sequência a ser inserido em um novo campo do pacote;
- O destinatário verifica esse número de sequência para determinar se o pacote recebido é ou não uma retransmissão;
- Para esse caso simples de protocolo pare-e-espere, um número de sequência de um bit é suficiente.

rdt2.1 - um protocolo melhorado para um canal com erros de bits (lado remetente)



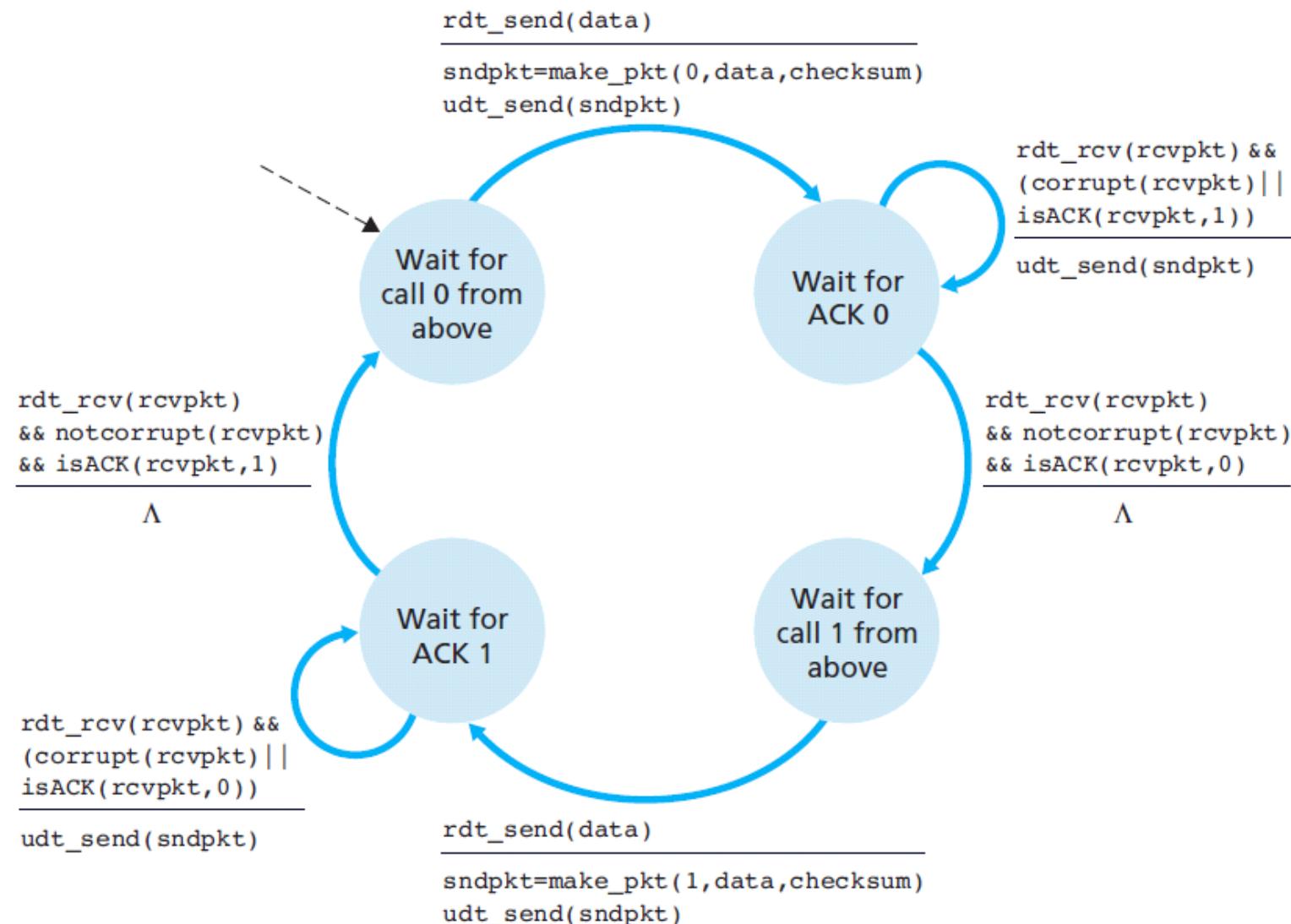
rdt2.1 - um protocolo melhorado para um canal com erros de bits (lado destinatário)



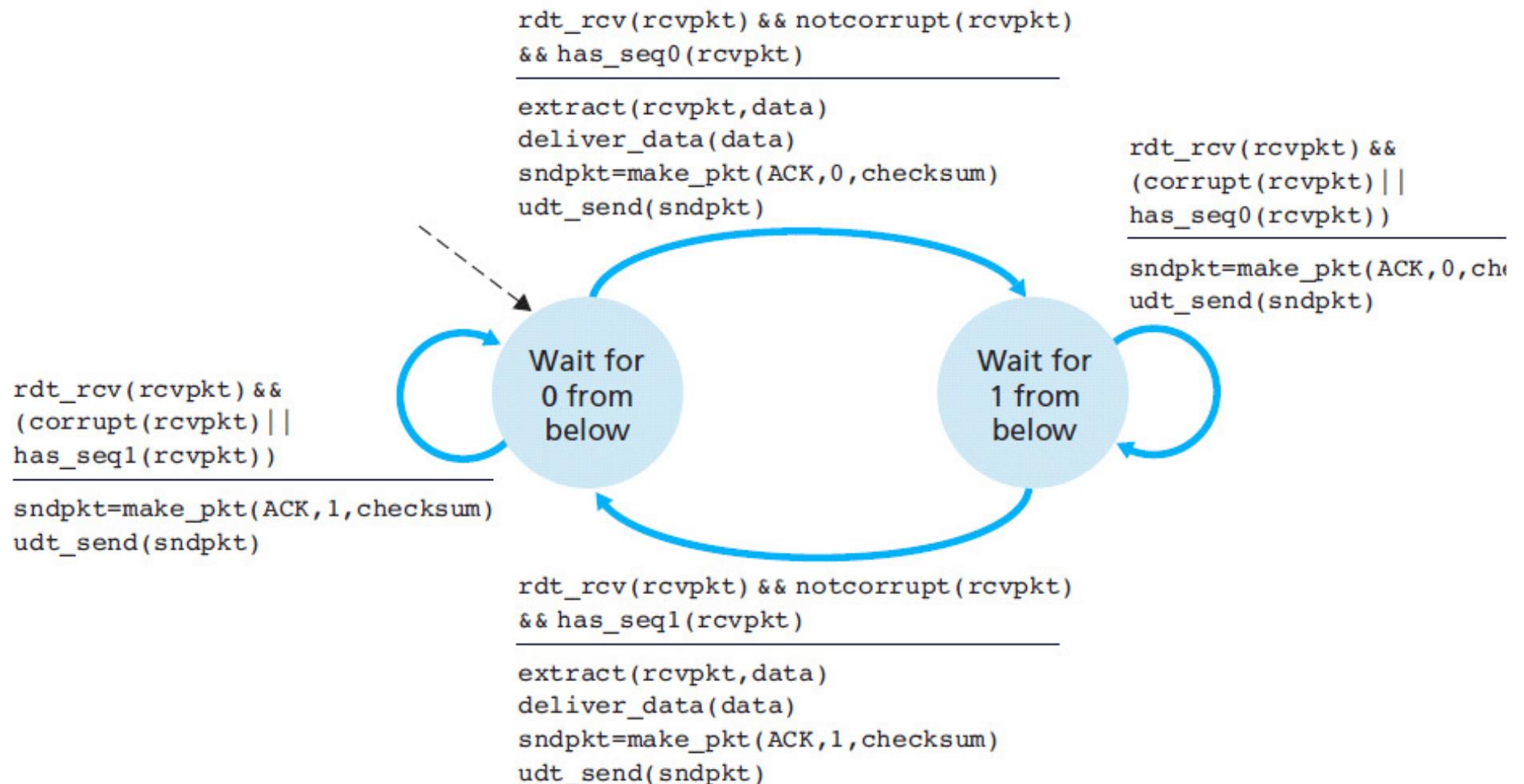
rdt2.2 - um protocolo melhorado para um canal com erros de bits (lado remetente)

- O protocolo rdt2.1 usa tanto o reconhecimento positivo como o negativo do destinatário ao remetente;
- Quando um pacote fora de ordem é recebido, o destinatário envia um reconhecimento positivo para o pacote que recebeu; quando um pacote corrompido é recebido, ele envia um reconhecimento negativo;
- Podemos conseguir o mesmo efeito de um pacote NAK se, em vez de enviamos um NAK, enviamos um ACK em seu lugar para o **último pacote corretamente recebido** - criação do **protocolo rdt2.2**;
- Um remetente que recebe dois ACKs para o mesmo pacote (**ACKS duplicados**) sabe que o destinatário não recebeu corretamente o pacote seguinte àquele para o qual estão sendo dados dois ACKs.

rdt2.2 - um protocolo melhorado para um canal com erros de bits (lado remetente)



rdt2.2 - um protocolo melhorado para um canal com erros de bits (lado destinatário)



Transferência Confiável de Dados

Como podemos tornar o protocolo rdt2.2 mais realista?

Transferência Confiável de Dados

Como podemos tornar o protocolo rdt2.2 mais realista?

Considerando que os pacotes podem ser perdidos no canal subjacente - criação do protocolo rdt3.0.

- O que fazer quando um pacote é perdido?
 - Técnicas já desenvolvidas em rdt2.2: Utilização de checksum, números de sequência, pacotes ACKs e retransmissões.

Como detectar a perda de um pacote?

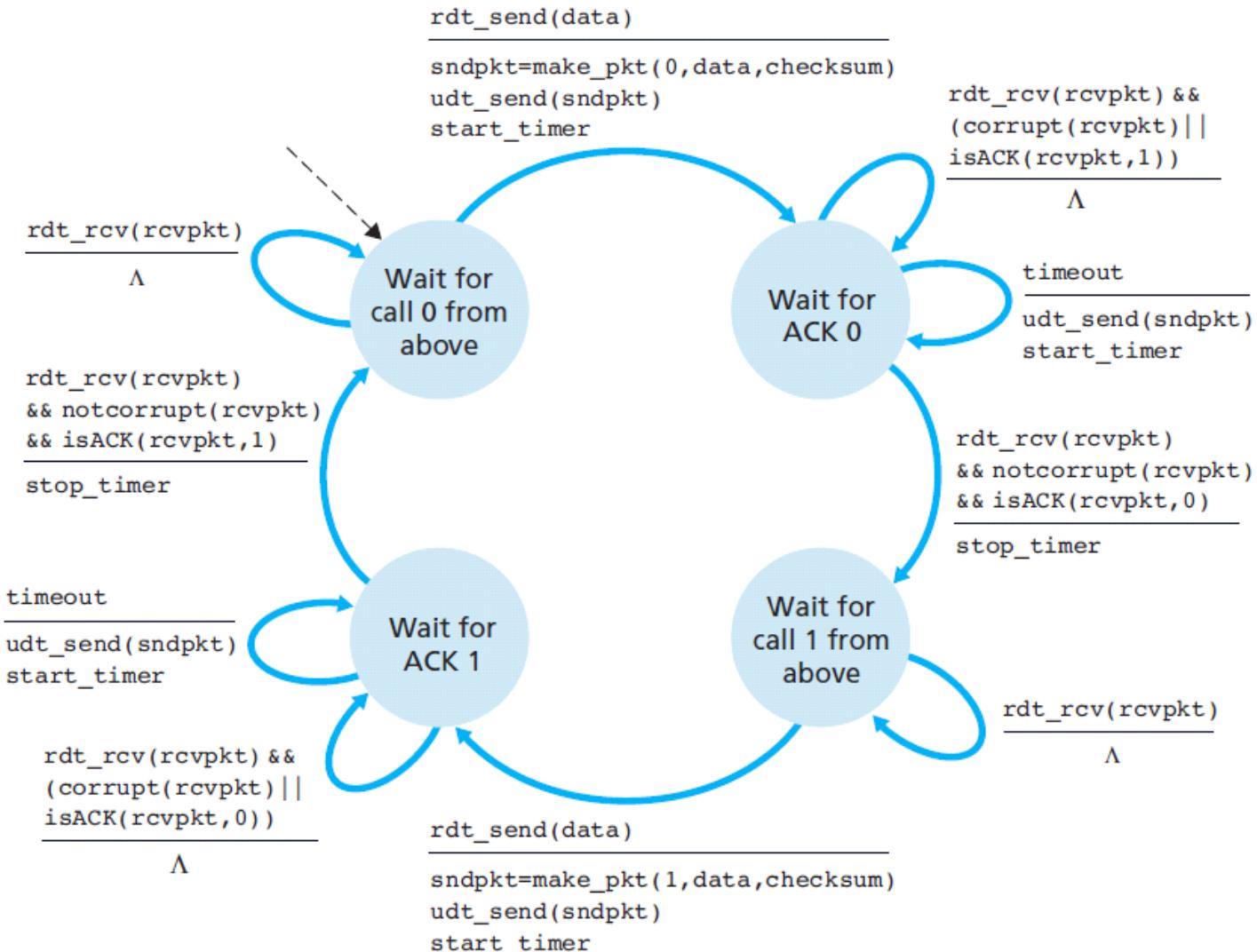
rdt3.0 - um protocolo para um canal com perda e com erros de bits

Como detectar a perda de um pacote?

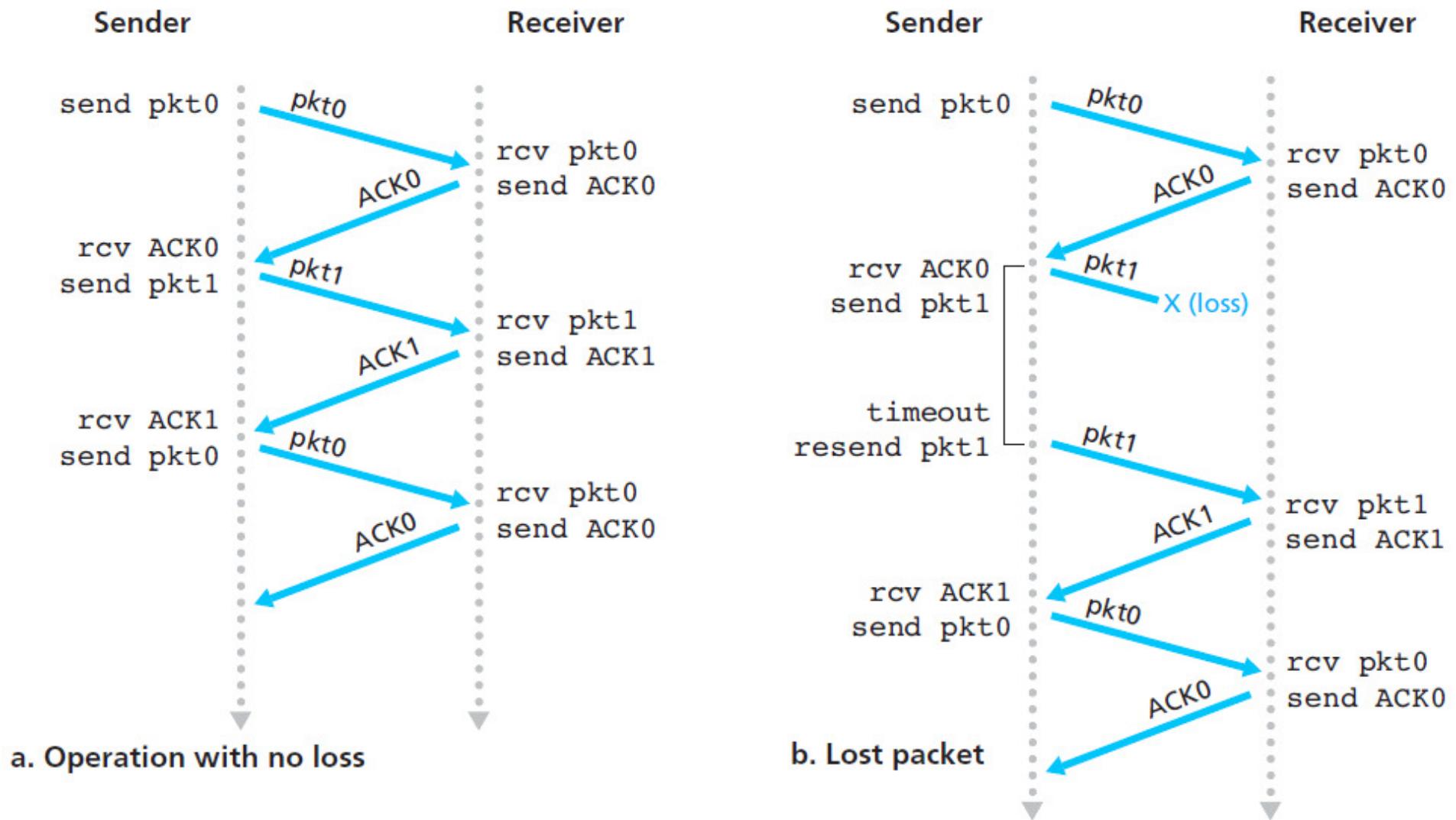
Utilizando um temporizador no remetente.

- Se um pacote do remetente ou um ACK do destinatário forem perdidos, **nenhuma resposta chegará** ao remetente vinda do destinatário;
- Se o remetente estiver disposto a **esperar o tempo suficiente para ter certeza** de que o pacote foi perdido, ele poderá simplesmente retransmitir o pacote de dados;
- Mas qual seria esse tempo?
 - **Atraso máximo** para o pior caso?
 - **Atraso provável**, mas não garantido?
- Se um pacote sofrer um atraso muito longo, existirão **pacotes de dados duplicados** no canal remetente-destinatário.

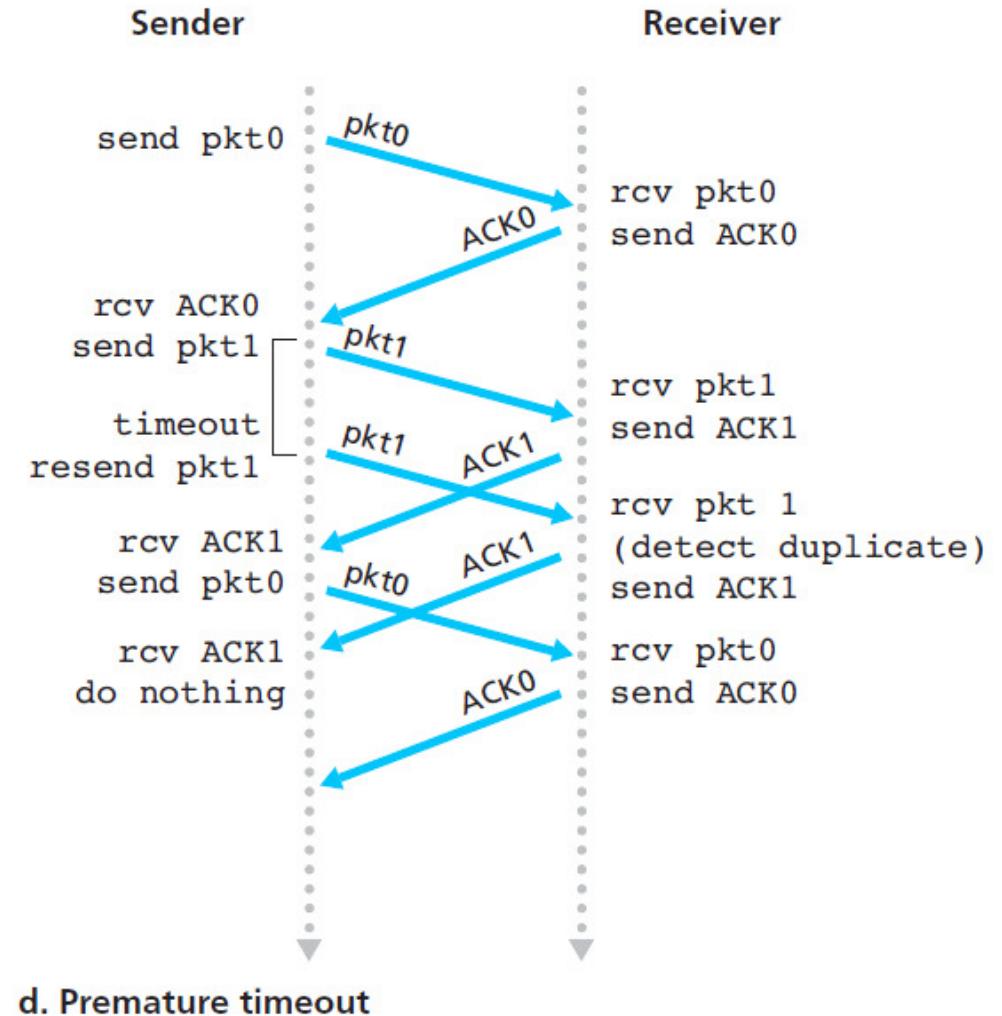
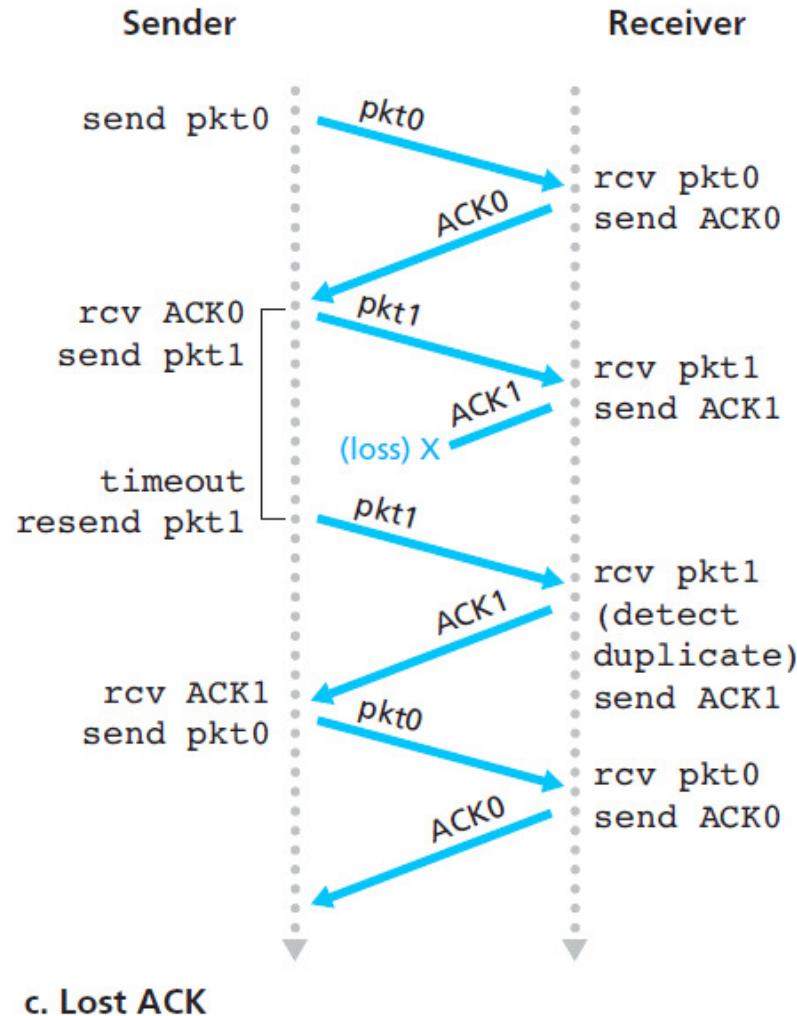
rdt3.0 - um protocolo para um canal com perda e com erros de bits (lado remetente)



Operação do rdt3.0 (protocolo bit alternante)



Operação do rdt3.0 (protocolo bit alternante)



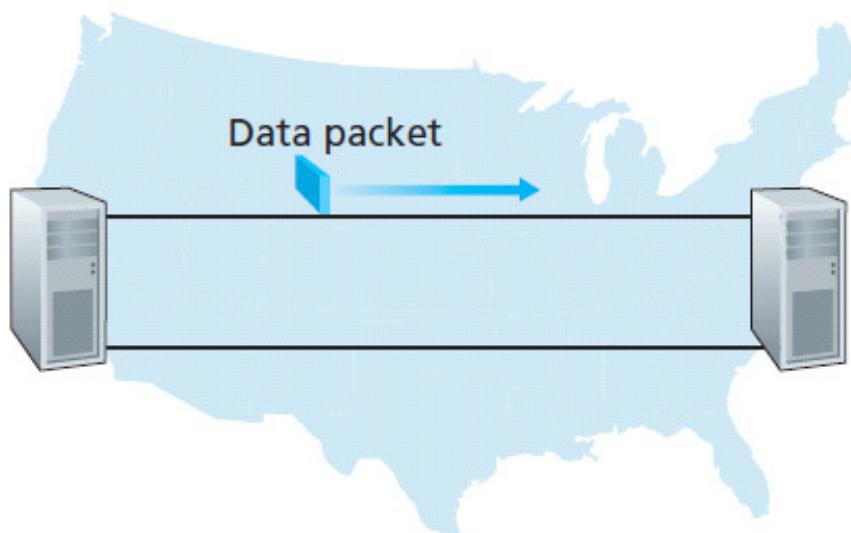
Protocolos de Transferência Confiável de Dados com Paralelismo

O protocolo rdt3.0 é eficiente em termos de ocupação da largura de banda?

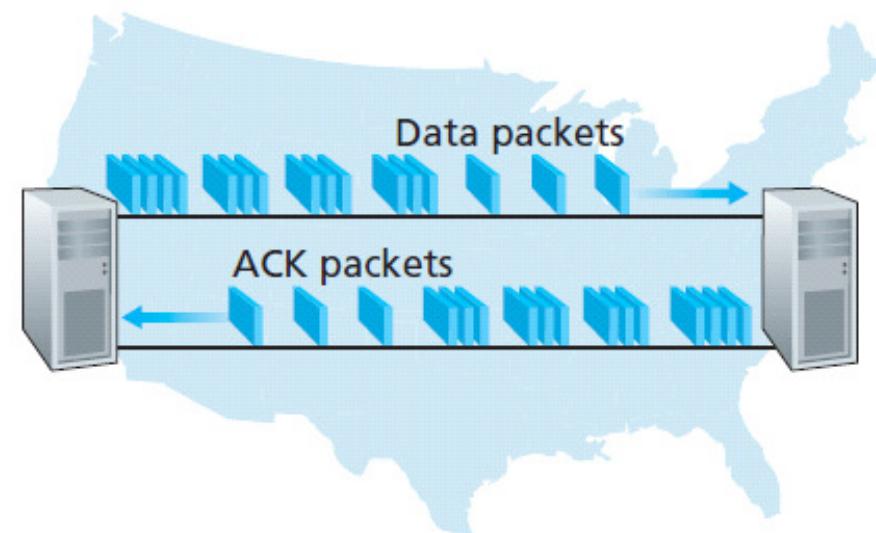
Protocolos de Transferência Confiável de Dados com Paralelismo

O protocolo rdt3.0 é eficiente em termos de ocupação da largura de banda?

Não, pois ele é um protocolo do tipo pare-e-espera.

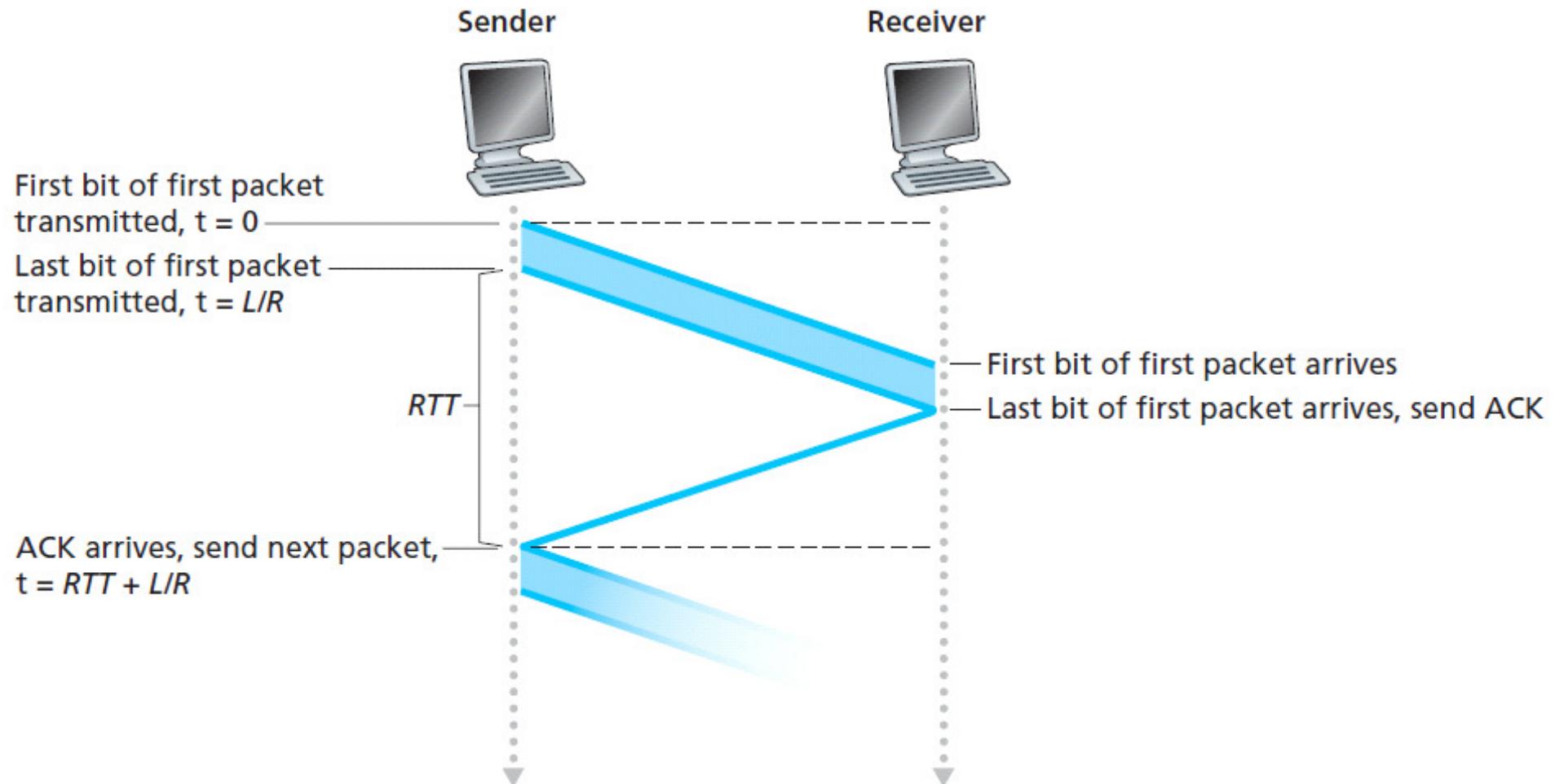


a. A stop-and-wait protocol in operation



b. A pipelined protocol in operation

Operação do protocolo pare-e-espere



Operação do protocolo pare-e-espere

- Exemplo: Dois hosts localizados em pontos opostos dos EUA querem se comunicar.
- O atraso de propagação de ida e volta é aproximadamente 30 ms;
- O canal de comunicação tem capacidade de transmissão de $R = 1 \text{ Gbps}$;
- Pacotes de tamanho $L = 1 \text{ Kbytes}$, incluindo dados e cabeçalho;
- Tempo necessário para realmente transmitir o pacote:

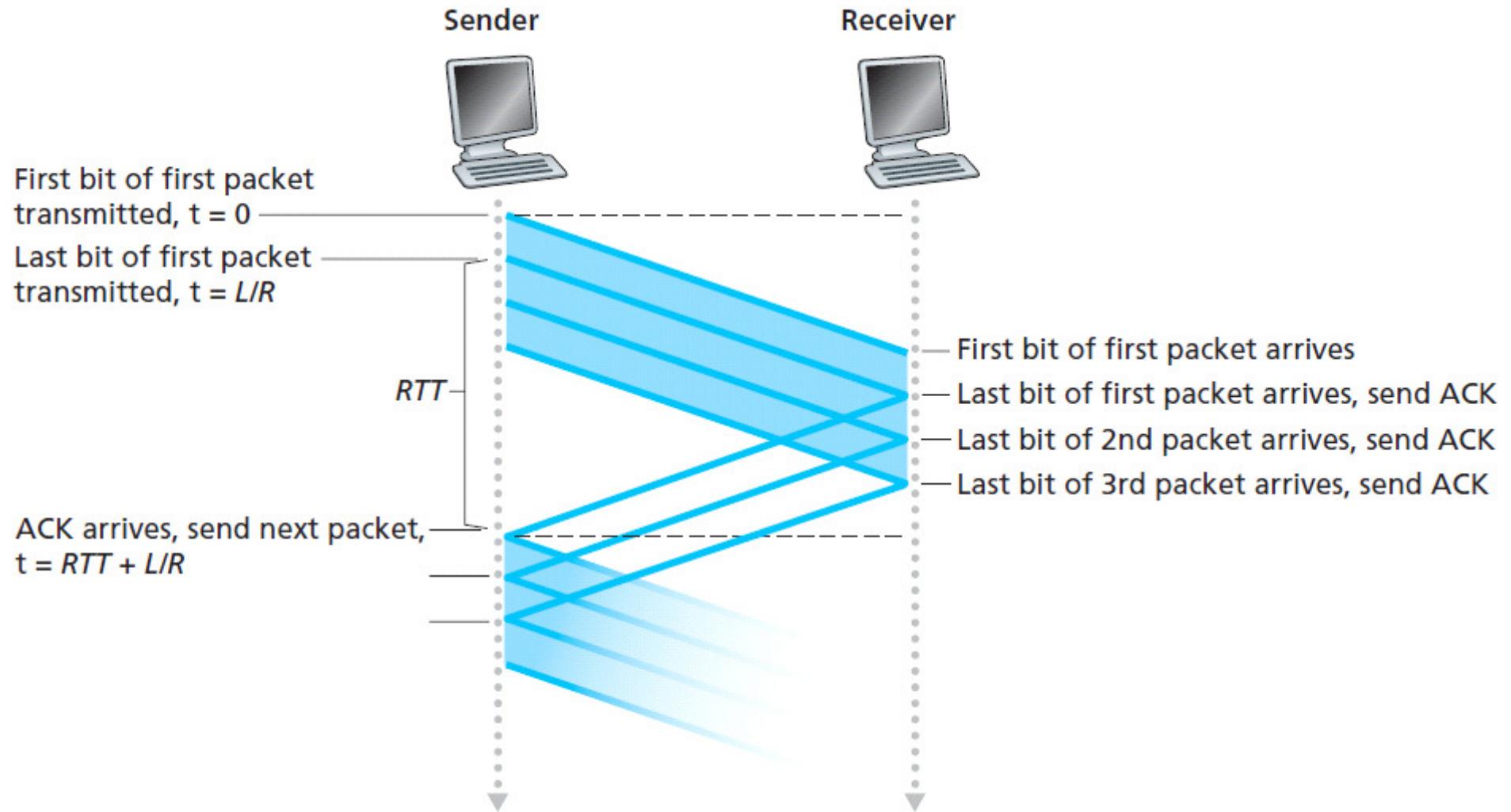
$$t_{trans} = \frac{L}{R} = \frac{1 \text{ Kbytes}}{1 \text{ Gbps}} = \frac{1 \times 10^3 \text{ bytes}}{1 \times 10^9 \text{ bytes/s}} = 8 \mu\text{s} \quad (1)$$

- Utilização do canal por parte do remetente:

$$U_{remet} = \frac{L/R}{RTT + L/R} = \frac{0,008}{30,008} = 0,00027 \quad (2)$$

- Vazão efetiva de apenas 267 kbps.

Operação do protocolo com paralelismo

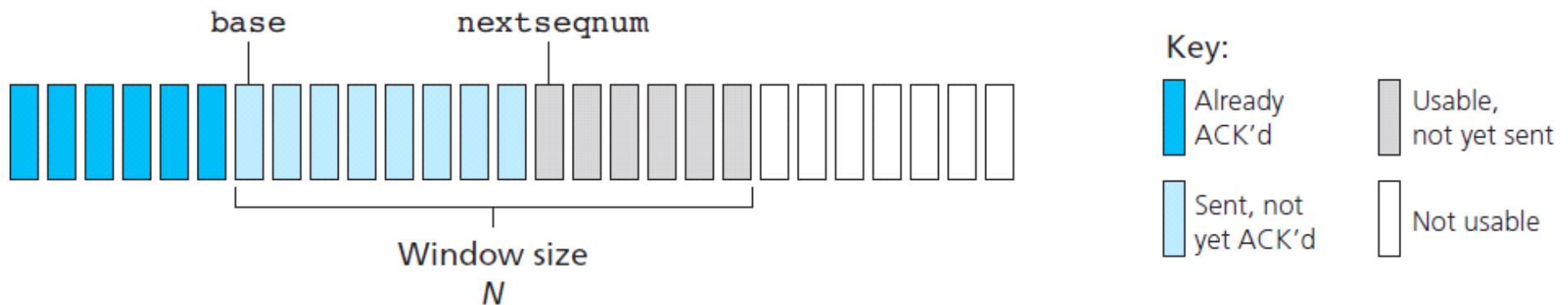


Operação do protocolo com paralelismo

- O paralelismo gera as seguintes consequências:
 - A faixa de números de sequência tem que ser **ampliada**, uma vez que pode haver vários pacotes não reconhecidos em trânsito;
 - Os lados remetente e destinatário dos protocolos podem ter de **reservar buffers** para mais de um pacote;
- Duas abordagens básicas em relação à recuperação de erros com paralelismo:
 - **Go-Back-N**;
 - **Repetição seletiva**.

Protocolo Go-Back-N (GBN)

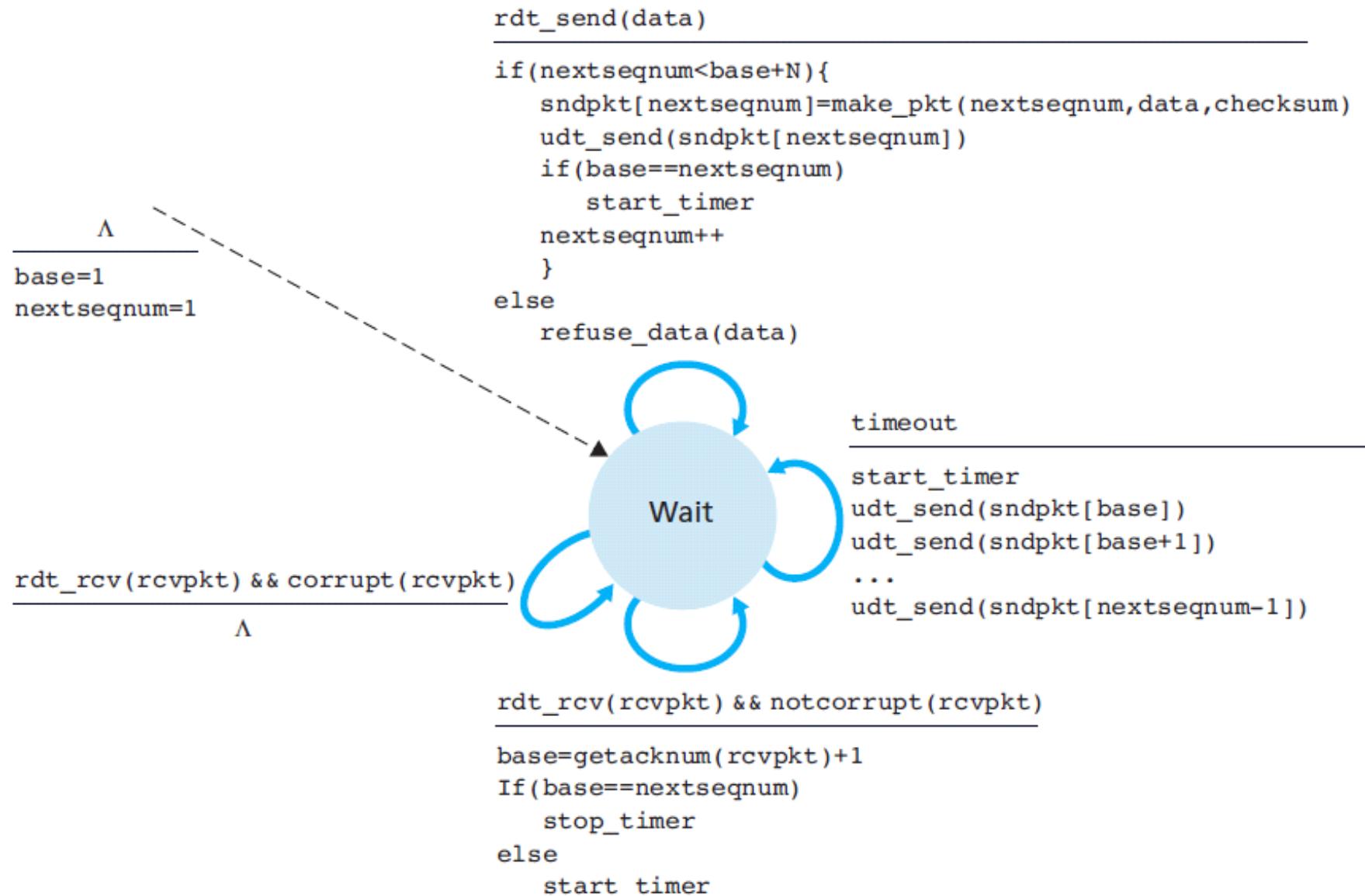
- O remetente é autorizado a transmitir múltiplos pacotes sem esperar por um reconhecimento, mas fica limitado a ter no máximo **N pacotes não reconhecidos**;
- O GBN é um **protocolo de janela deslizante**, onde N é o **tamanho da janela**;
- Se k for o número de bits no campo de número de sequência de pacote, a faixa de números de sequência será então $[0, 2^k - 1]$.



Protocolo Go-Back-N (GBN)

- O remetente GBN deve responder a três tipos de eventos:
 - **Chamada vinda de cima**;
 - **Recebimento de um ACK**: reconhecimento **cumulativo**;
 - **Um esgotamento de temporização**: o remetente reenvia **todos** os pacotes que tinham sido previamente enviados mas que ainda não tinham sido reconhecidos.
- O destinatário GBN **descarta** os pacotes que chegam fora de ordem.

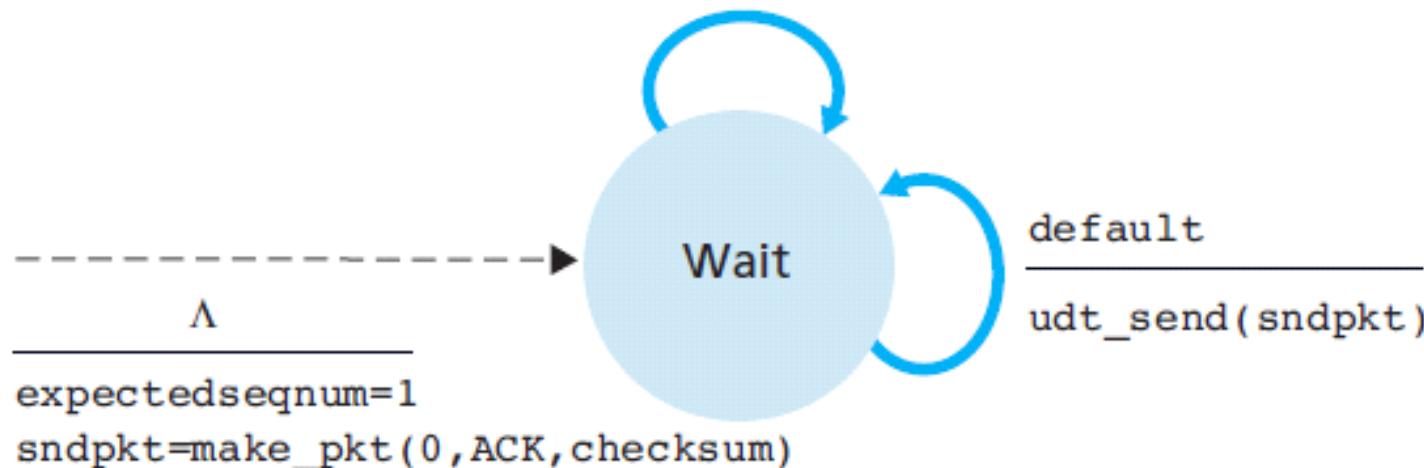
Descrição do remetente Go-Back-N



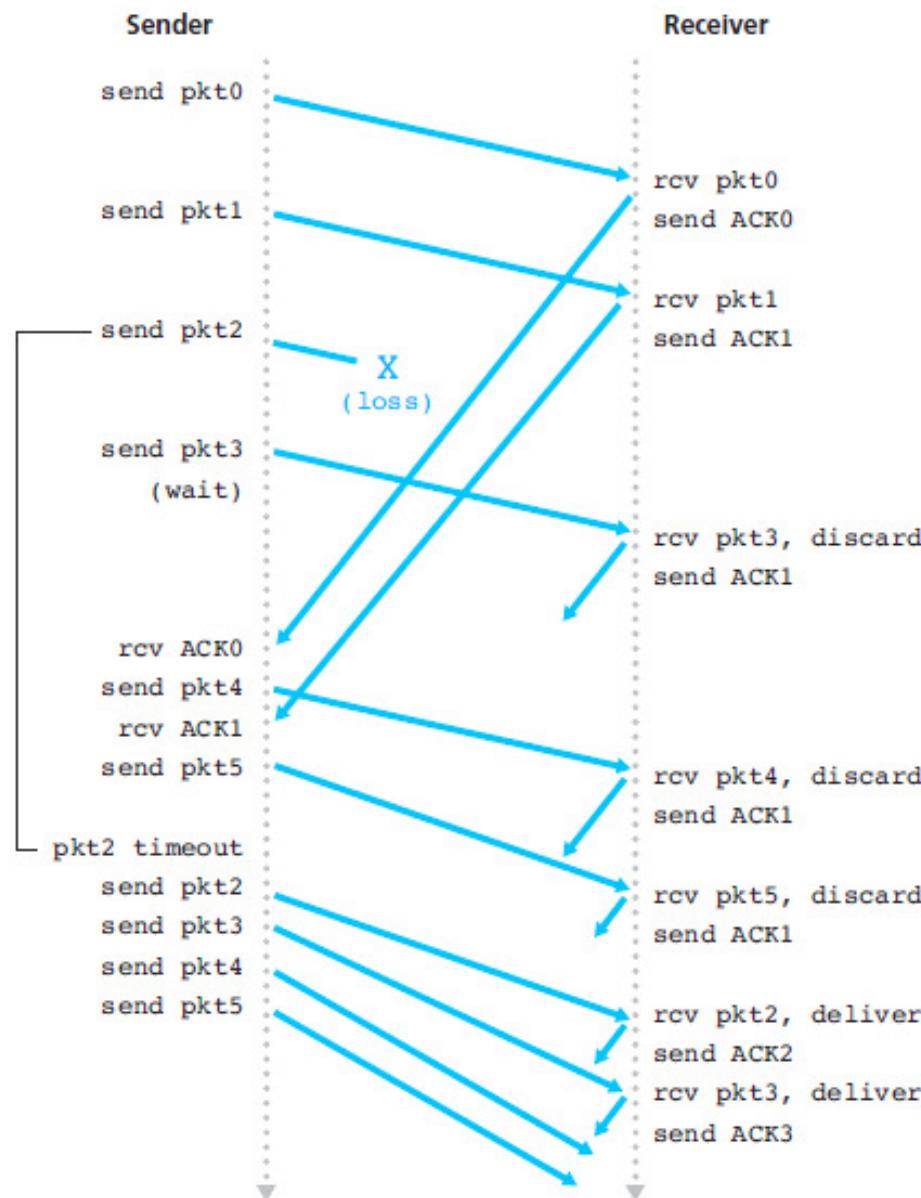
Descrição do destinatário Go-Back-N

```
rdt_rcv(rcvpkt)
  && notcorrupt(rcvpkt)
  && hasseqnum(rcvpkt, expectedseqnum)
```

```
extract(rcvpkt, data)
deliver_data(data)
sndpkt=make_pkt(expectedseqnum, ACK, checksum)
udt_send(sndpkt)
expectedseqnum++
```



Operação do Go-Back-N



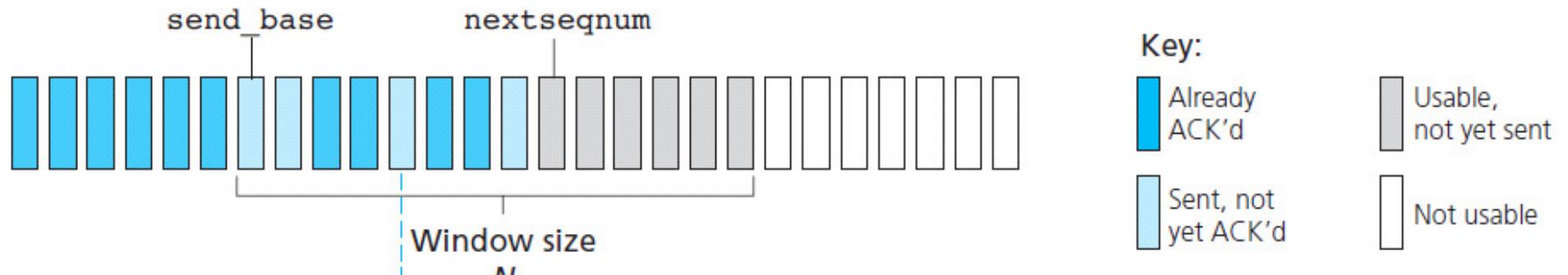
Protocolo de Repetição Seletiva (SR)

- O protocolo GBN sofre com **problemas de desempenho** quando o tamanho da janela e o produto entre o atraso e a largura de banda são grandes;
- Nesse caso podem haver **muitos pacotes pendentes** na rede, pois um único erro de pacote pode fazer com que o GBN retransmita um grande número de pacotes, muitos deles desnecessariamente;
- Protocolos SR evitam **retransmissões desnecessárias** porque fazem o remetente retransmitir somente os pacotes suspeitos de terem sido recebidos com erro no destinatário;

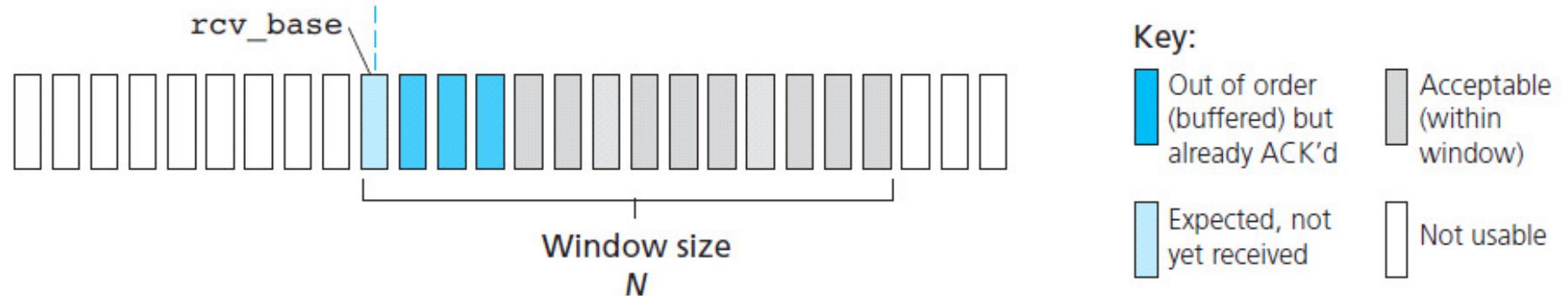
Protocolo de Repetição Seletiva (SR)

- Essa retransmissão inicial, somente quando necessária, exige que o destinatário reconheça **individualmente** os pacotes recebidos de modo correto;
- Ao contrário do GBN, o SR remetente já terá recebido ACKs para alguns dos pacotes da janela;
- O SR destinatário reconhecerá um pacote corretamente recebido esteja ele ou não na ordem certa; pacotes fora de ordem ficam no **buffer** até que todos os pacotes faltantes (com número de sequências menores) sejam recebidos.

Visões que os protocolos SR remetente e destinatário têm do espaço de número de sequência



a. Sender view of sequence numbers



b. Receiver view of sequence numbers

Eventos e ações do protocolo SR remetente

- **Dados recebidos de cima**

- O SR remetente verifica o próximo número de sequência disponível para o pacote;
- Se o número de sequência está dentro da **janela do remetente**, os dados são empacotados e enviados;
- Caso contrário, eles são armazenados ou devolvidos à camada superior para transmissão posterior.

- **Esgotamento de temporização**

- **Proteção** contra perda de pacotes;
- Cada pacote deve ter seu **próprio temporizador** lógico, já que apenas um pacote será transmitido quando a temporização se esgotar.

Eventos e ações do protocolo SR remetente

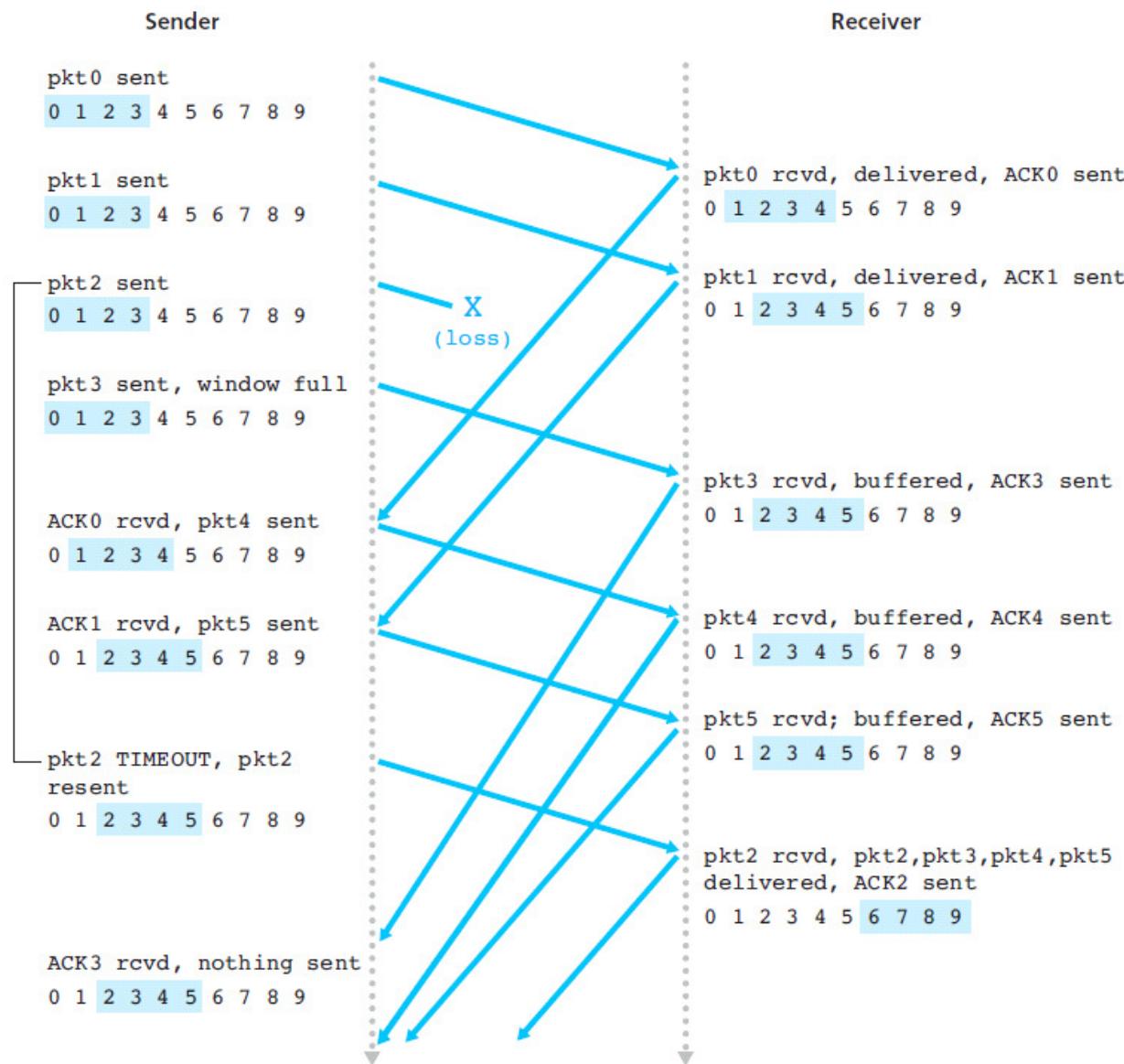
- **ACK recebido**

- O SR remetente marcará aquele pacote como recebido, contanto que esteja na janela;
- Se o número de sequência do pacote for igual a sendbase, a base da janela se deslocará para a frente até o pacote não reconhecido que tiver o menor número de sequência;
- Se a janela se deslocar e houver pacotes não transmitidos com números de sequência que agora caem dentro da janela, esses pacotes são transmitidos.

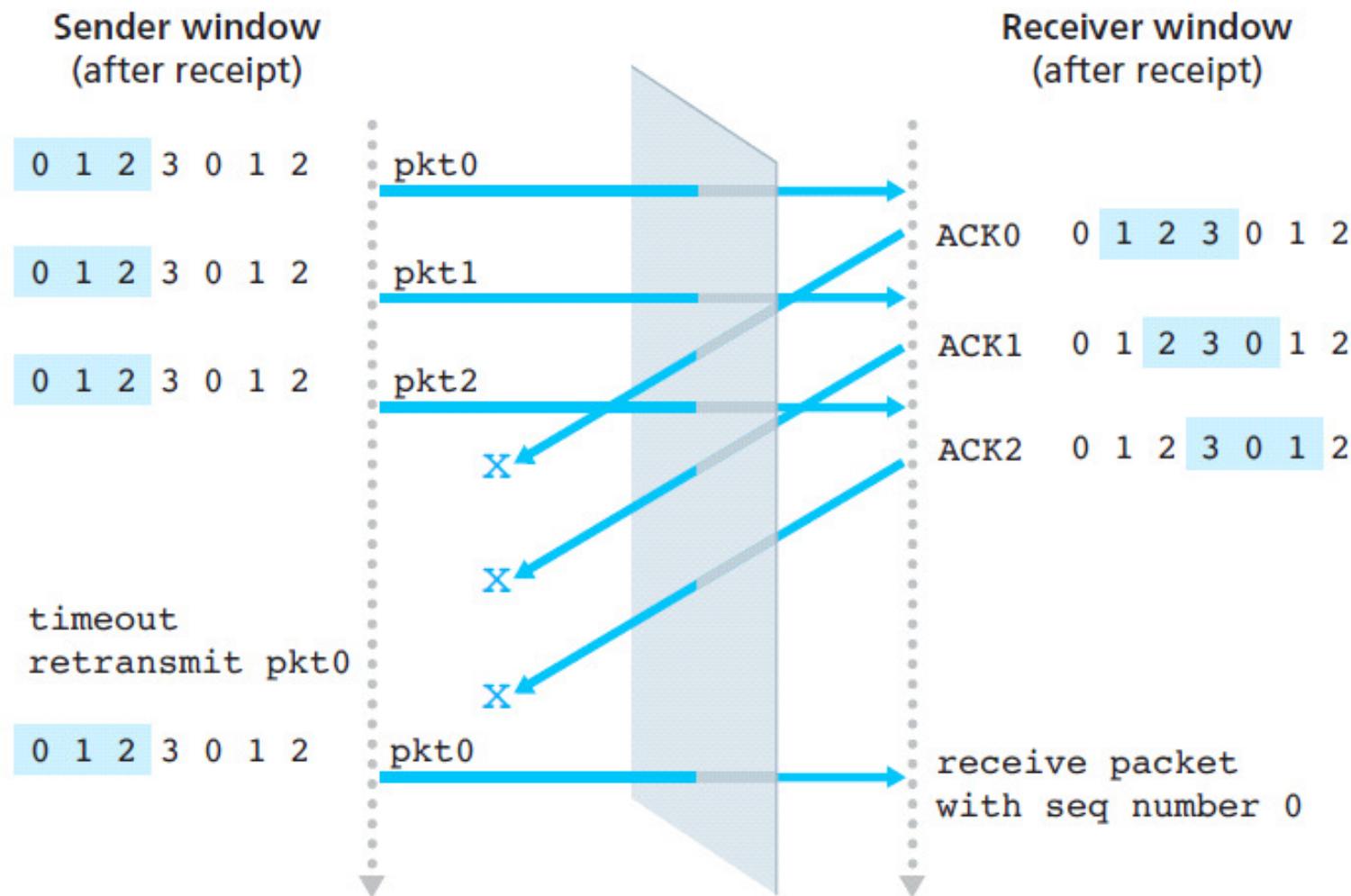
Eventos e ações do protocolo SR destinatário

- Pacote com número de sequência no intervalo $[rcvbase, rcvbase+N-1]$ foi corretamente recebido
 - O pacote recebido cai dentro da **janela do destinatário** e um pacote ACK seletivo é devolvido ao remetente;
 - Se o pacote não tiver sido recebido anteriormente, irá para o buffer;
 - A janela destinatária é então deslocada para a frente de acordo com o número de pacotes entregues à camada superior.
- Pacote com número de sequência no intervalo $[rcvbase-N, rcvbase-1]$ é recebido
 - Um ACK deve ser gerado mesmo que esse pacote já tenha sido reconhecido anteriormente pelo destinatário.
- Qualquer outro evento
 - Ignore o pacote.

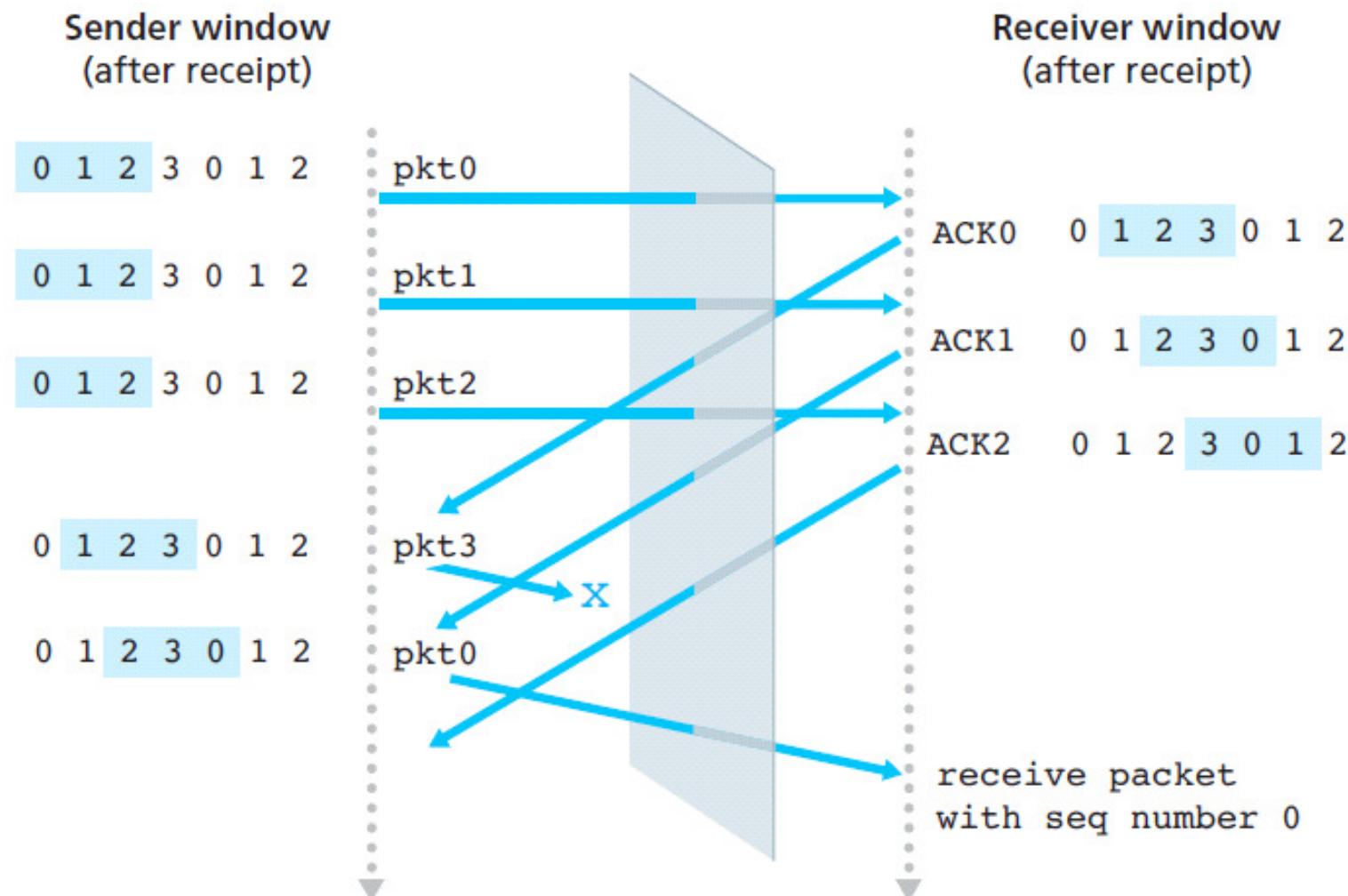
Operação do protocolo SR



Dilema do remetente SR com janelas muito grandes: um novo pacote ou uma retransmissão?



Dilema do remetente SR com janelas muito grandes: um novo pacote ou uma retransmissão?

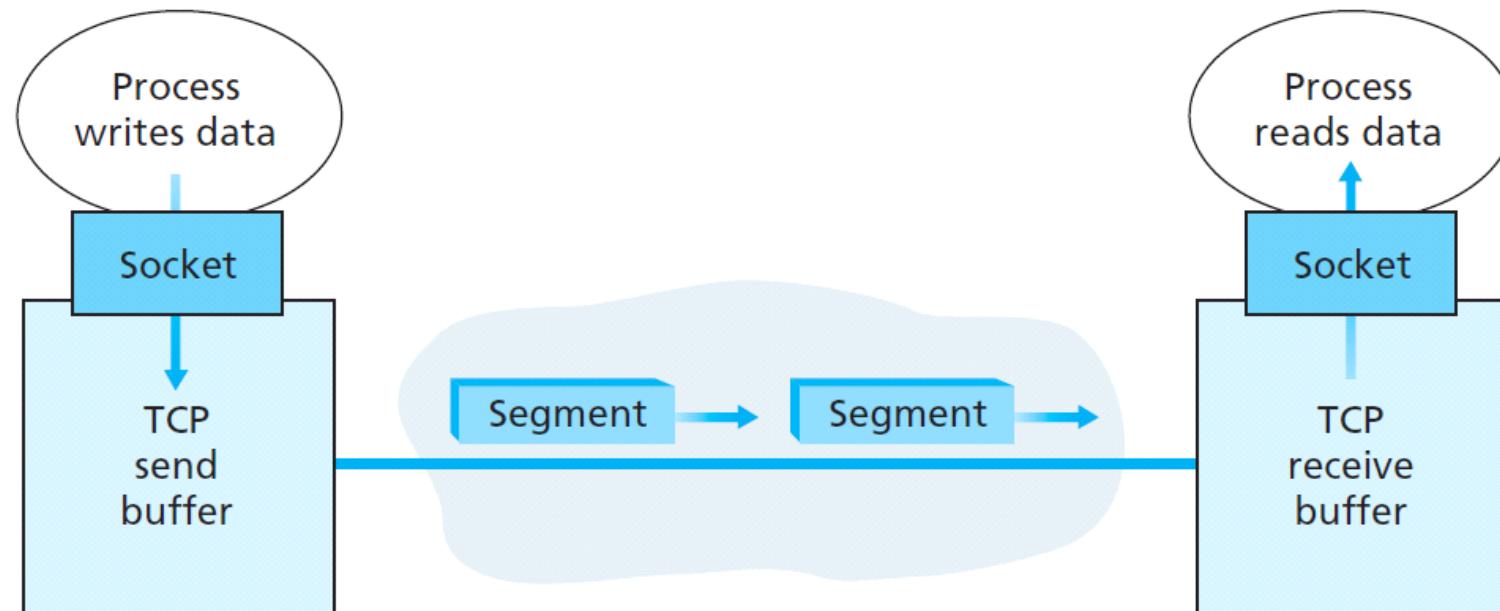


A conexão TCP

- Protocolo **orientado para conexão**: antes que um processo de aplicação possa começar a enviar dados a outro, os dois processos precisam primeiramente se “apresentar” um ao outro;
- Uma conexão TCP não é um circuito TDM ou FDM fim a fim, como acontece em uma rede de comutação de circuitos; é uma **conexão lógica** entre hosts finais;
- Uma conexão TCP provê um **serviço full-duplex**;
- Uma conexão TCP é sempre **ponto a ponto**, i.e., entre um único remetente e um único destinatário;
- O procedimento de estabelecimento de conexão do TCP é frequentemente denominado **apresentação de três vias** (*3-way handshake*);

A conexão TCP

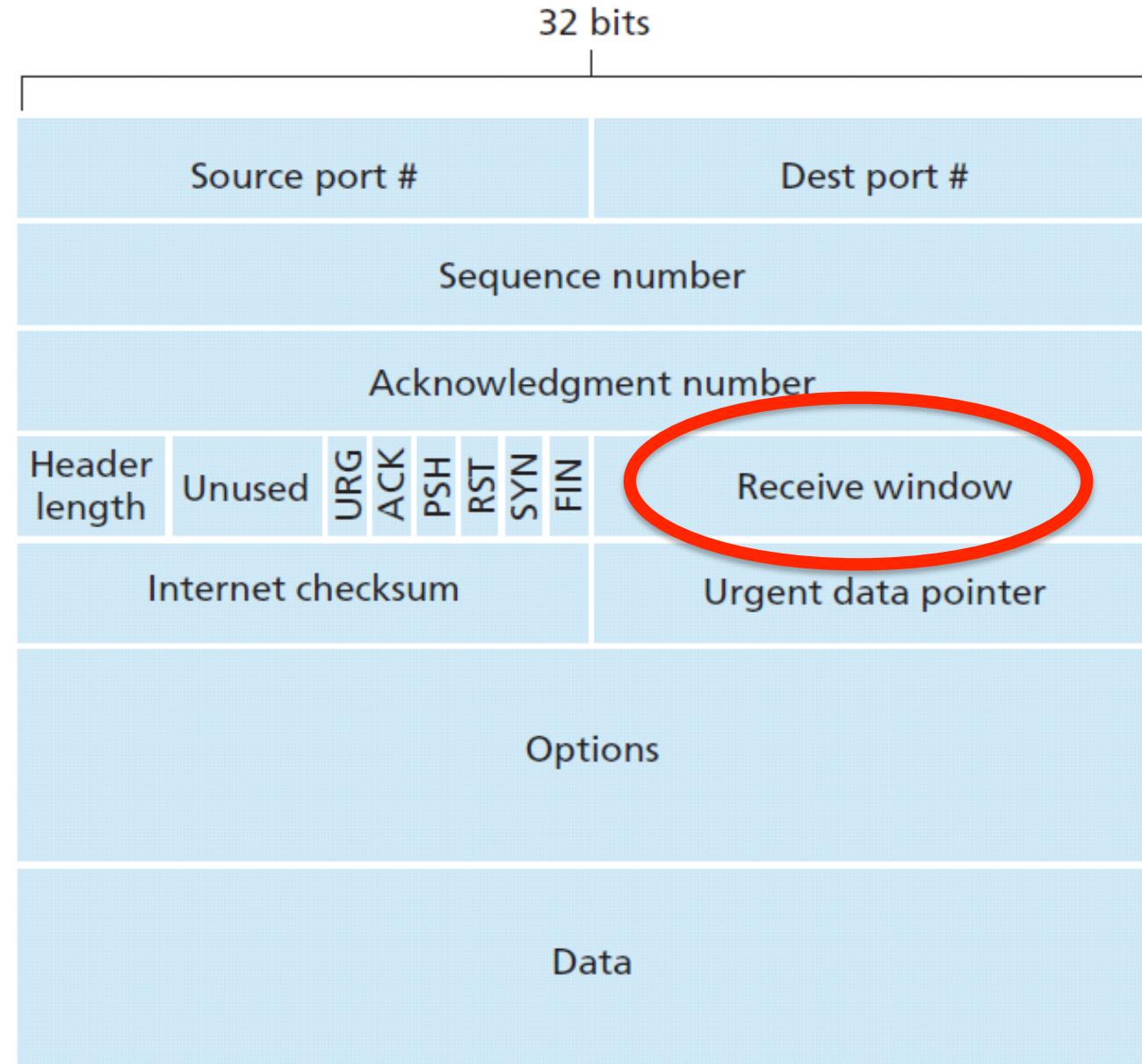
- O TCP cliente direciona seus dados para o **buffer de envio** da conexão, que é um dos buffers reservados durante a apresentação de 3 vias inicial;
- A quantidade máxima de dados de aplicação que pode ser retirada e colocada em um segmento é limitada pelo **tamanho máximo do segmento** (maximum segment size - MSS);



A conexão TCP

- Primeiramente se determina o tamanho do maior quadro de camada de enlace que pode ser enviado pelo host remetente: **unidade máxima de transmissão** (maximum transmission unit - MTU);
- Em seguida, se estabelece um MSS que garanta que um segmento TCP (quando encapsulado em um datagrama IP) caberá em um único quadro de camada de enlace;
- Valores comuns de MTU são 1460 bytes, 536 bytes e 512 bytes;
- O TCP combina cada porção de dados do cliente com um cabeçalho TCP, formando, assim, **segmentos TCP**.

Estrutura do segmento TCP



Números de sequência e números de reconhecimento

Números de sequência

O número de sequência para um segmento é o número do primeiro byte do segmento.

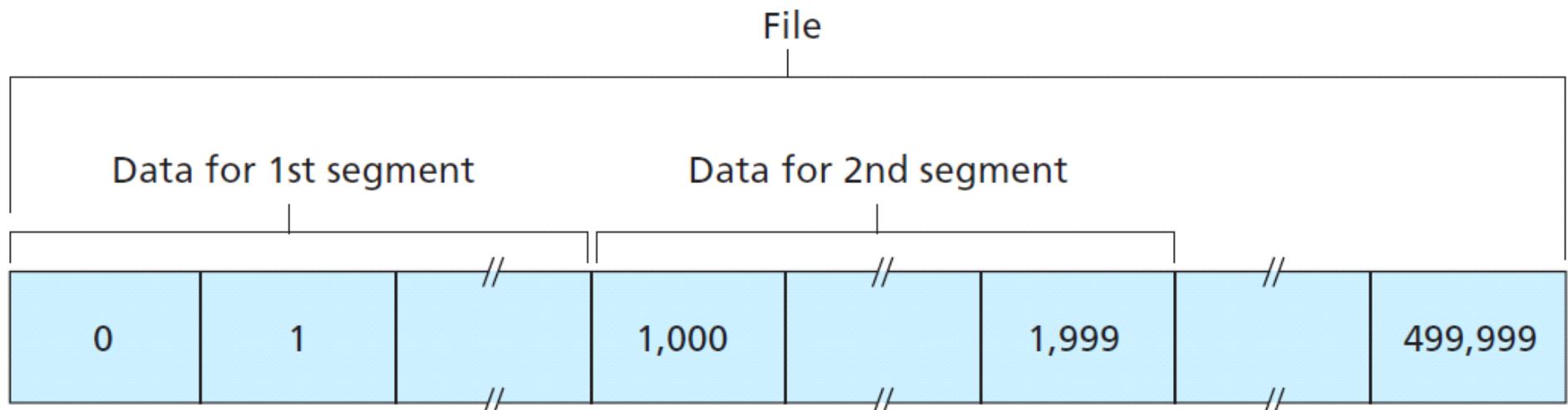
Números de reconhecimento

O número de reconhecimento que o host A atribui a seu segmento é o número de sequência do próximo byte que ele estiver aguardando do host B.

Números de sequência e números de reconhecimento

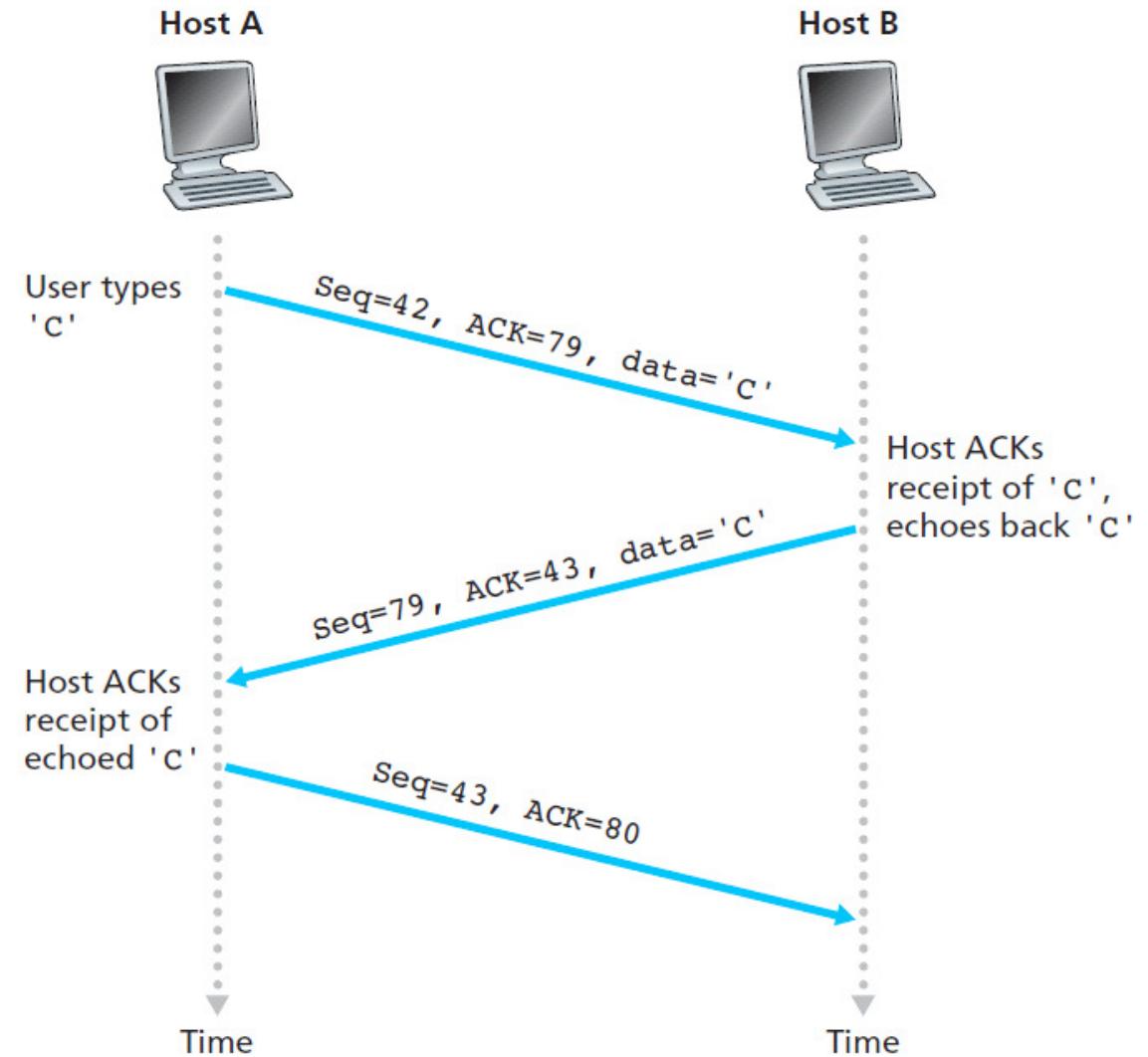
- **Exemplo:**

- O host A quer enviar um arquivo de 500 Kbytes para o host B;
- O MSS é 1 Kbyte;
- É atribuído o número 0 para o primeiro byte da cadeia de dados.

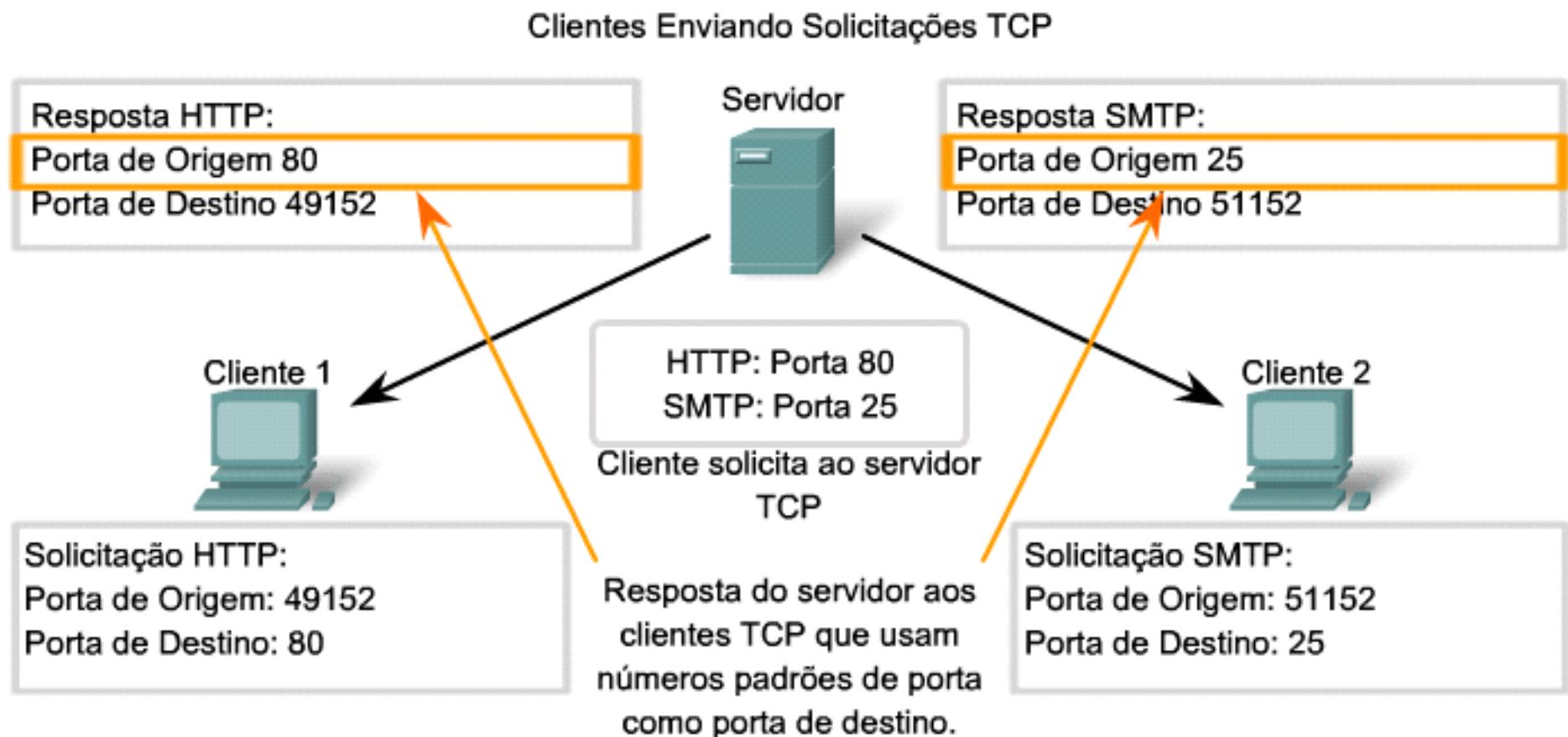


Números de sequência e números de reconhecimento

- Exemplo: Telnet
- Números de sequência iniciais: 42 (cliente) e 79 (servidor);
- O reconhecimento para dados do cliente para o servidor **pegou uma carona** no segmento de dados do servidor para o cliente: *piggyback*.



Processos TCP em Servidores



Estimativa de RTT

- O TCP utiliza um mecanismo de **controle de temporização/retransmissão** para recuperar segmentos perdidos;
- A duração do temporizador deve ser maior do que o tempo de ida e volta da conexão (RTT);
- O RTT para um segmento (*SampleRTT*) é a quantidade de tempo transcorrido entre o momento em que o segmento é enviado e o momento em que é recebido um reconhecimento para o segmento.

Estimativa de RTT

- Filtragem do RTT utilizando uma **média móvel exponencial ponderada**:

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT} \quad (3)$$

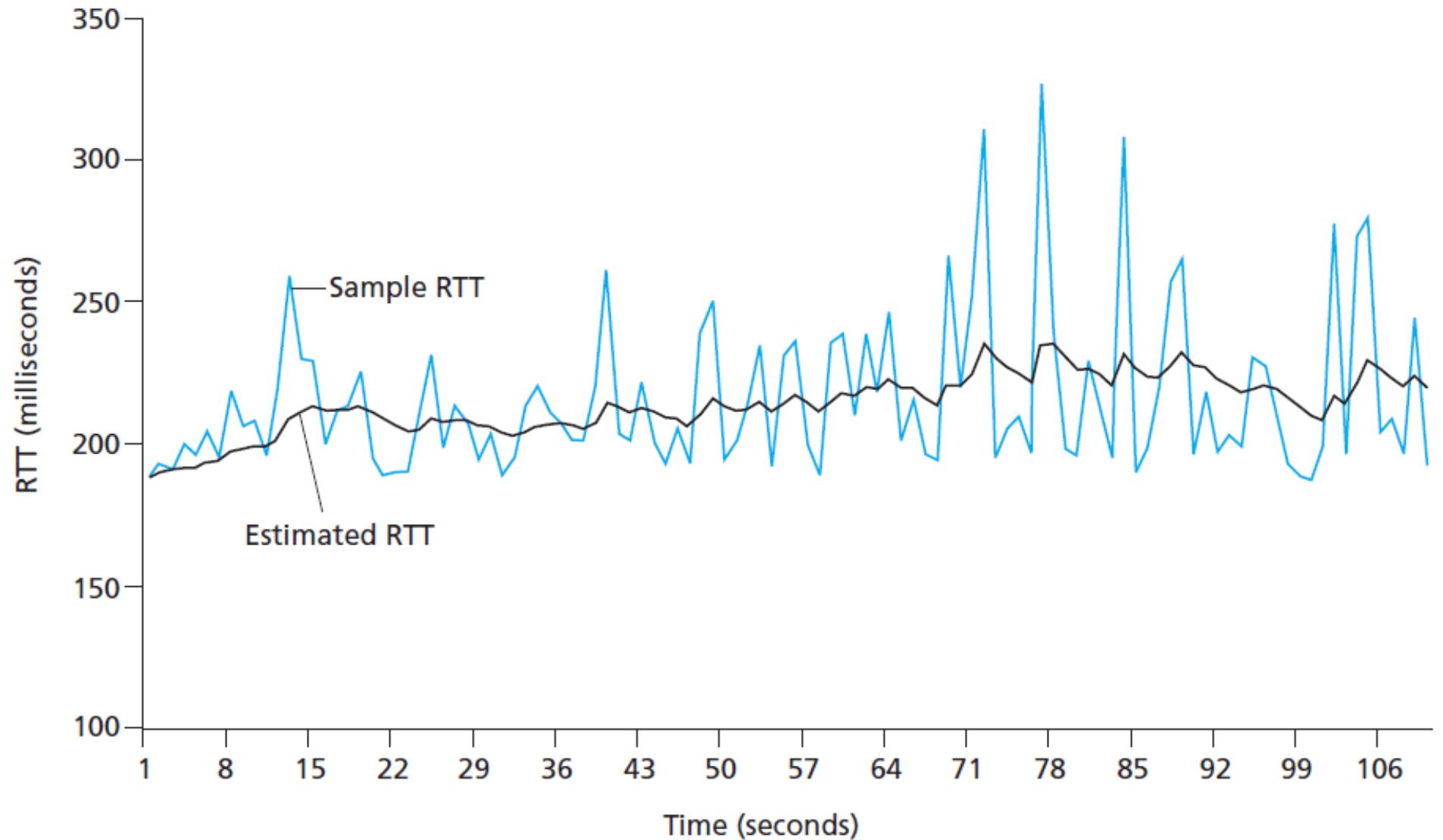
- Estimativa do **desvio típico** entre SampleRTT e EstimatedRTT :

$$\text{DevRTT} = (1 - \beta) \cdot \text{DevRTT} + \beta \cdot |\text{SampleRTT} - \text{EstimatedRTT}| \quad (4)$$

- Temporização de retransmissão:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT} \quad (5)$$

Estimativa do RTT



Transferência Confiável de Dados

- O TCP cria um serviço de **transferência confiável** de dados sobre o serviço de melhor esforço do IP (não confiável);
- O TCP utiliza apenas um **único temporizador** de retransmissão, mesmo que haja vários segmentos transmitidos ainda não reconhecidos;
- O único temporizador está associado com o **mais antigo** segmento não reconhecido;
- Há três eventos importantes relacionados com a transmissão e retransmissão de dados no TCP remetente:
 - ➊ Dados recebidos da aplicação acima;
 - ➋ Esgotamento do temporizador;
 - ➌ Recebimento de ACK.

Transferência Confiável de Dados

- **Dados recebidos da aplicação acima:**

```
NextSeqNum = InitialSeqNumber
SendBase = InitialSeqNumber
create TCP segment with sequence number NextSeqNum
if (timer currently not running) start timer
pass segment to IP
NextSeqNum = NextSeqNum + length(data)
```

Transferência Confiável de Dados

- **Esgotamento do temporizador:**

retransmit not-yet-acknowledged segment with smallest sequence number
start timer

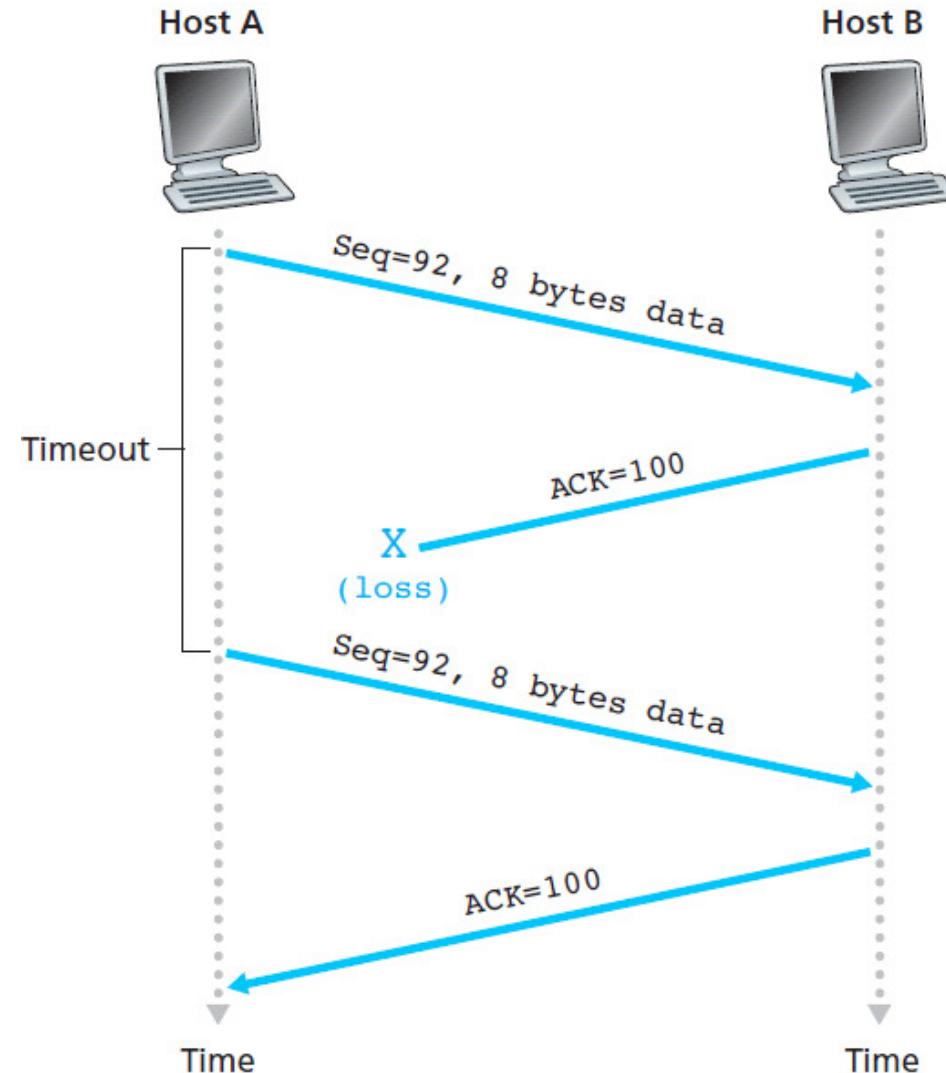
Transferência Confiável de Dados

- **recebimento de ACK, com o valor do campo ACK igual a y:**

```
if (y > SendBase)
SendBase = y
if (there are currently any not-yet-acknowledged
segments)
start timer
```

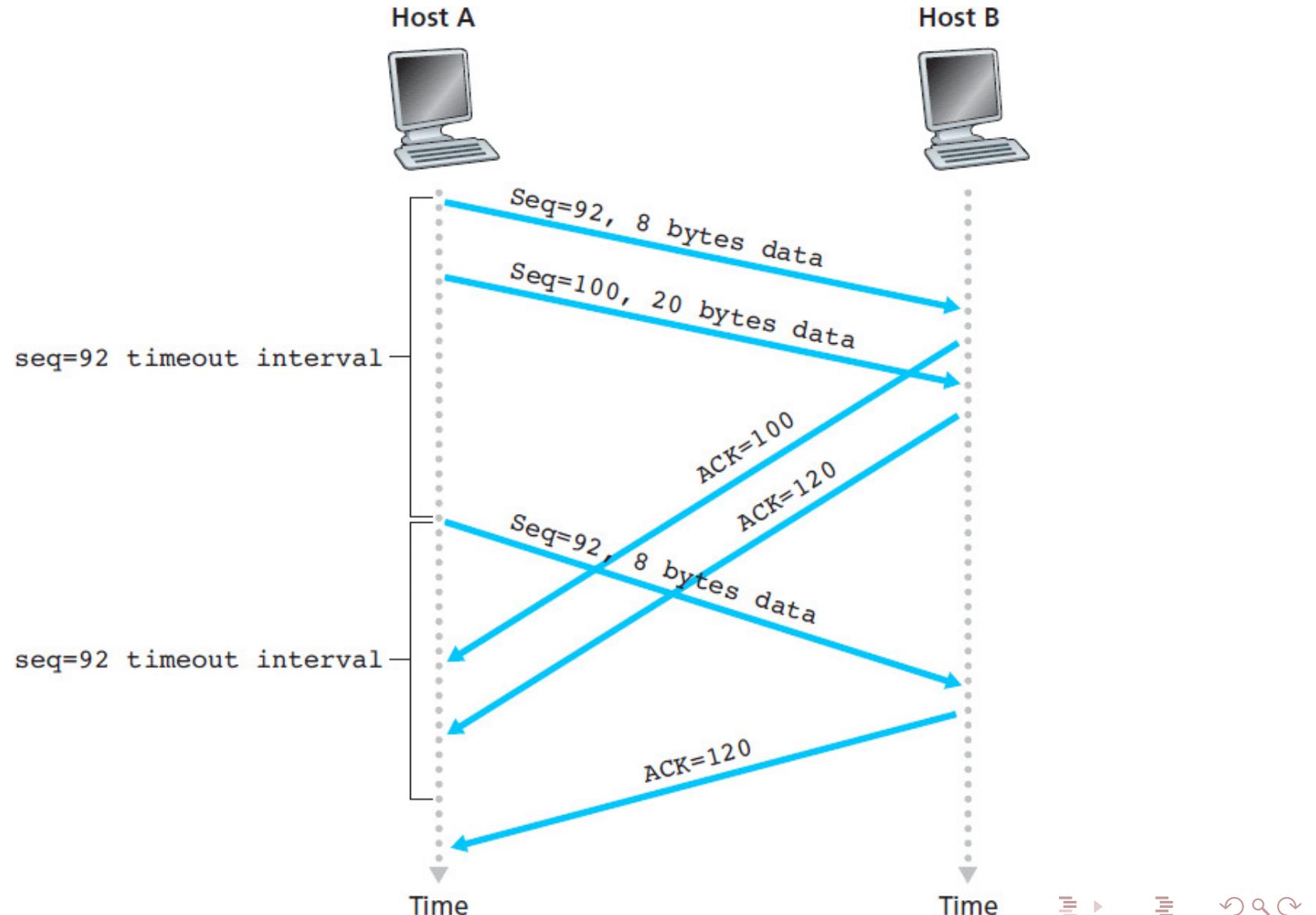
Alguns Cenários Interessantes

Cenário 1: Retransmissão devido a um reconhecimento perdido



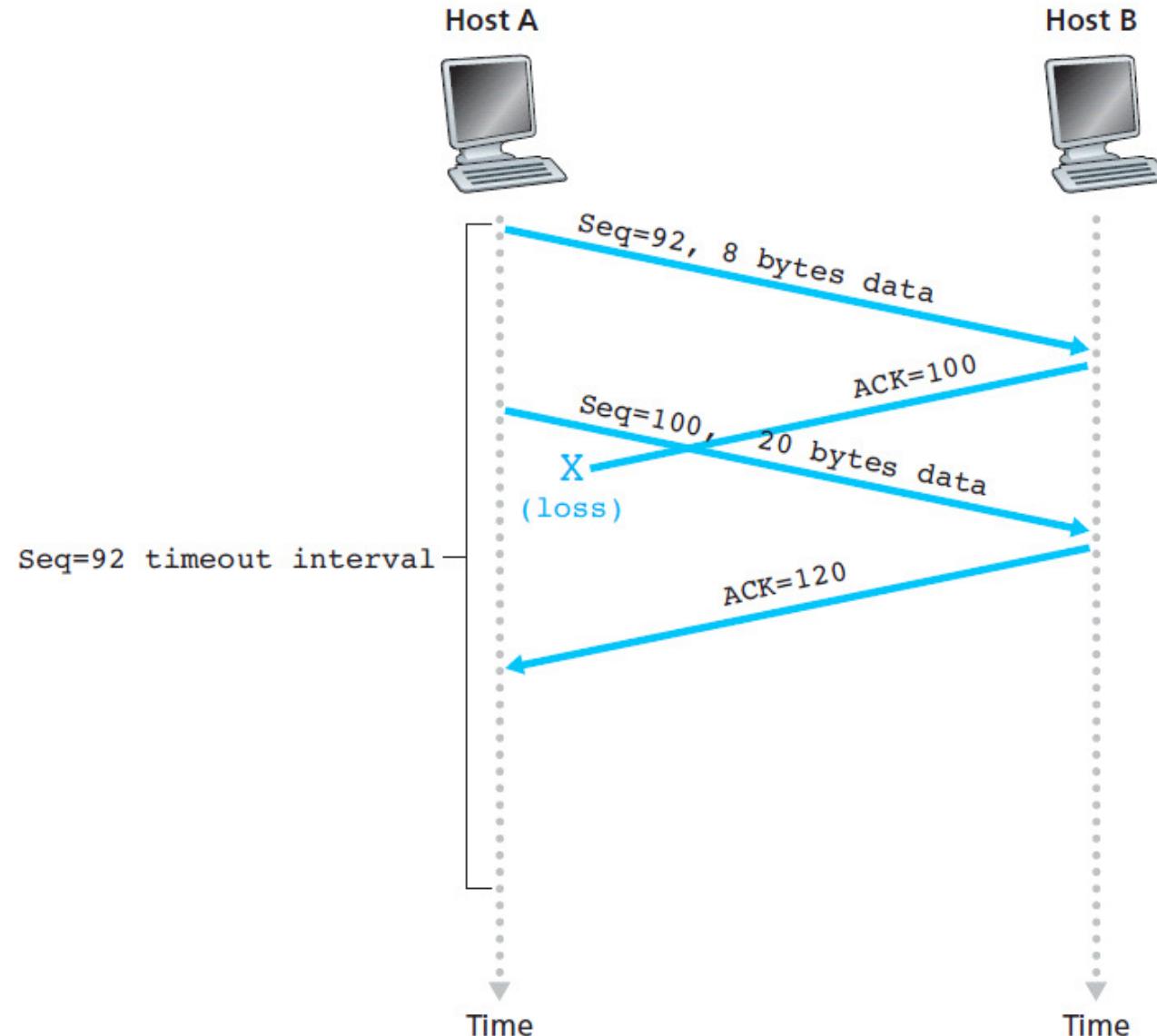
Alguns Cenários Interessantes

Cenário 2: Segmento 100 não retransmitido



Alguns Cenários Interessantes

Cenário 3: Um reconhecimento cumulativo evita retransmissão do primeiro segmento



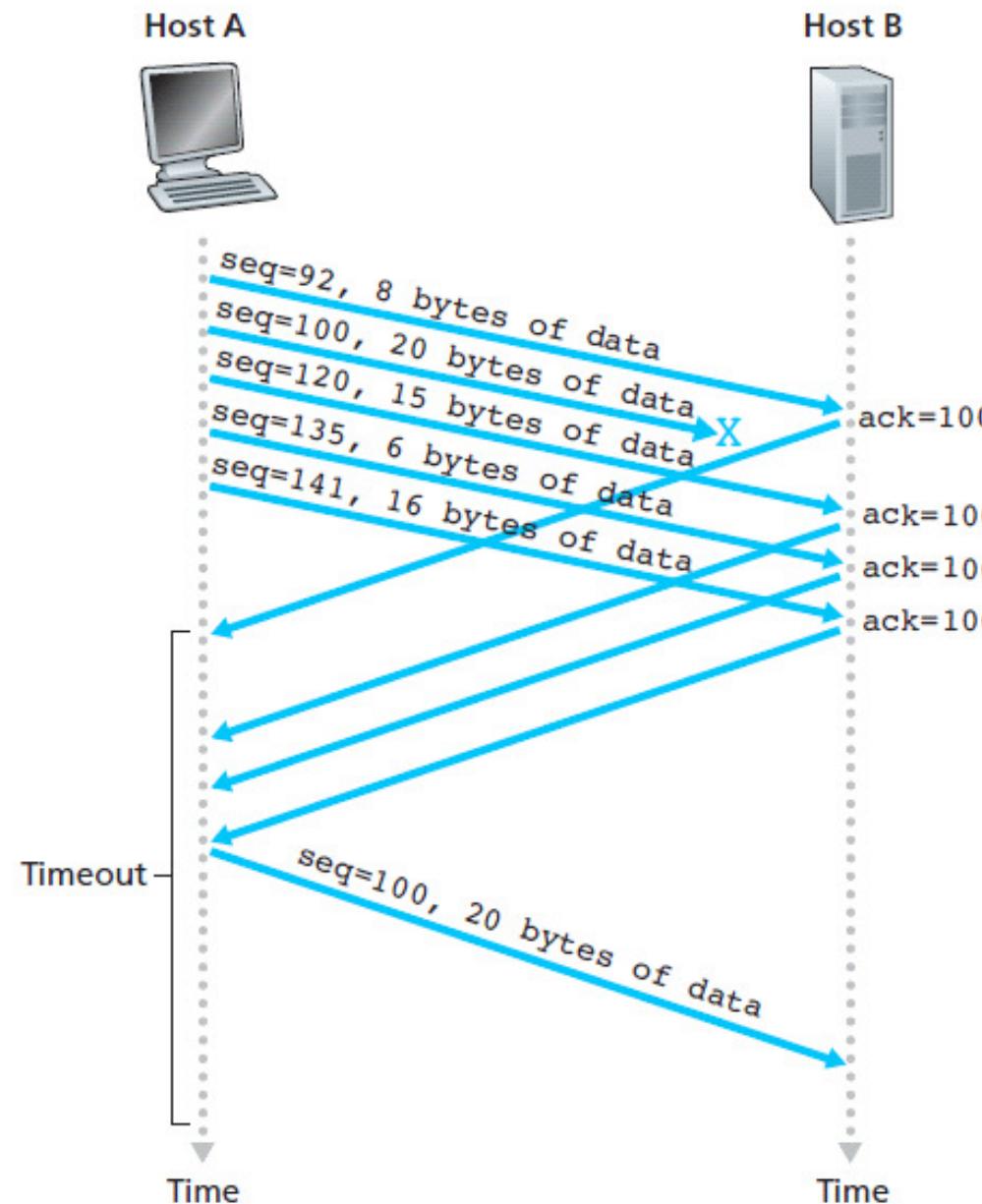
Duplicação do tempo de expiração

- Sempre que ocorre o evento de expiração do temporizador, o TCP retransmite o segmento ainda não reconhecido que tenha o **menor número de sequência**;
- A cada retransmissão, o TCP ajusta o próximo tempo de expiração para o **dobro do valor anterior** em vez de derivá-los dos últimos *EstimatedRTT* e *DevRTT*;
- O tempo de expiração aumentará **exponencialmente** a cada nova retransmissão;
- Sempre que o temporizador é reiniciado após qualquer um dos outros dois eventos (dados recebidos da aplicação ou recebimento de ACK), o *TimeoutInterval* será derivado dos valores mais recentes de *EstimatedRTT* e *DevRTT*.

Retransmissão rápida

- Um dos problemas de retransmissões acionadas por expiração de temporizador é que o período de expiração pode ser **relativamente longo**;
- O remetente pode com frequência detectar perda de pacote bem antes de ocorrer o evento de expiração, observando os **ACKs duplicados**;
- Um ACK duplicado é um ACK que reconhece novamente um segmento para o qual o remetente já recebeu um reconhecimento anterior;
- Quando o destinatário reconhece uma **lacuna na corrente de dados**, ele simplesmente reconhece novamente o último byte de dados que recebeu na ordem;
- No caso de receber três ACKs duplicados, o remetente realiza uma **retransmissão rápida**, retransmitindo o segmento que falta **antes** da expiração do temporizador do segmento.

Retransmissão Rápida



Retransmissão Rápida

Event	TCP Receiver Action
Arrival of in-order segment with expected sequence number. All data up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 msec for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK transmission.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. Gap detected.	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Go-Back-N ou repetição seletiva?

O TCP é um protocolo GBN ou SR?

Go-Back-N ou repetição seletiva?

O TCP é um protocolo GBN ou SR?

Ele é um híbrido dos dois.

Quais as semelhanças e diferenças entre o TCP e o GBN?

Go-Back-N ou repetição seletiva?

Quais as semelhanças e diferenças entre o TCP e o GBN?

- **Semelhanças com o GBN**

- Os reconhecimentos são cumulativos;
- Segmentos recebidos fora de ordem não são reconhecidos individualmente pelo destinatário;
- O remetente precisa somente atualizar os valores de SendBase e NextSeqNum.

- **Diferenças com o GBN**

- O TCP armazena segmentos recebidos fora de ordem;
- Em caso de perda do pacote $n < N$, o GBN retransmitiria não somente o pacote n , mas também todos os pacotes subsequentes, mesmo que já estivessem reconhecidos;
- O TCP retransmitiria no máximo o segmento n , e nem o faria se ele recebesse o reconhecimento do segmento $n + 1$ antes do timeout do pacote n .

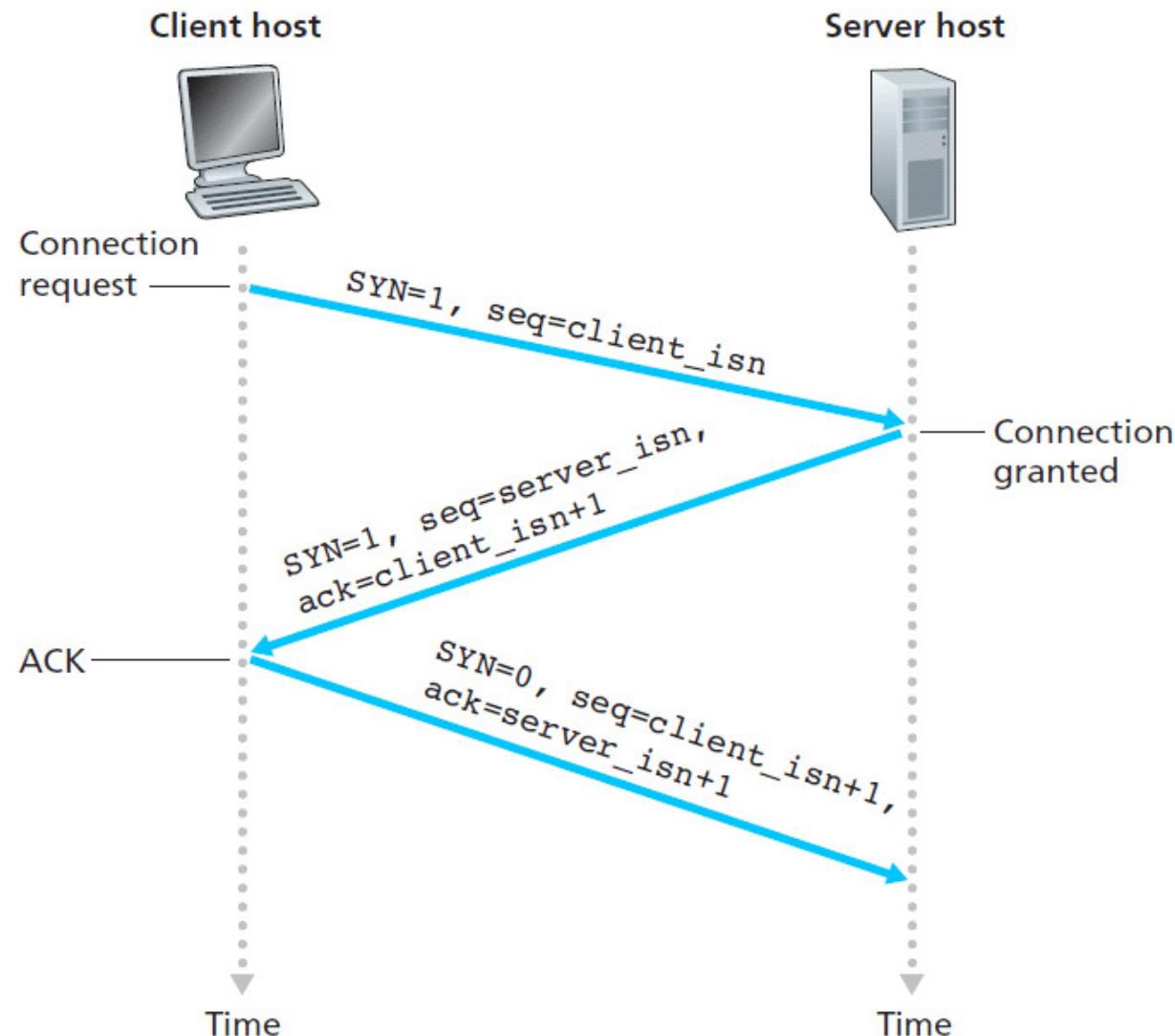
Go-Back-N ou repetição seletiva?

Quais as semelhanças e diferenças entre o TCP e o SR?

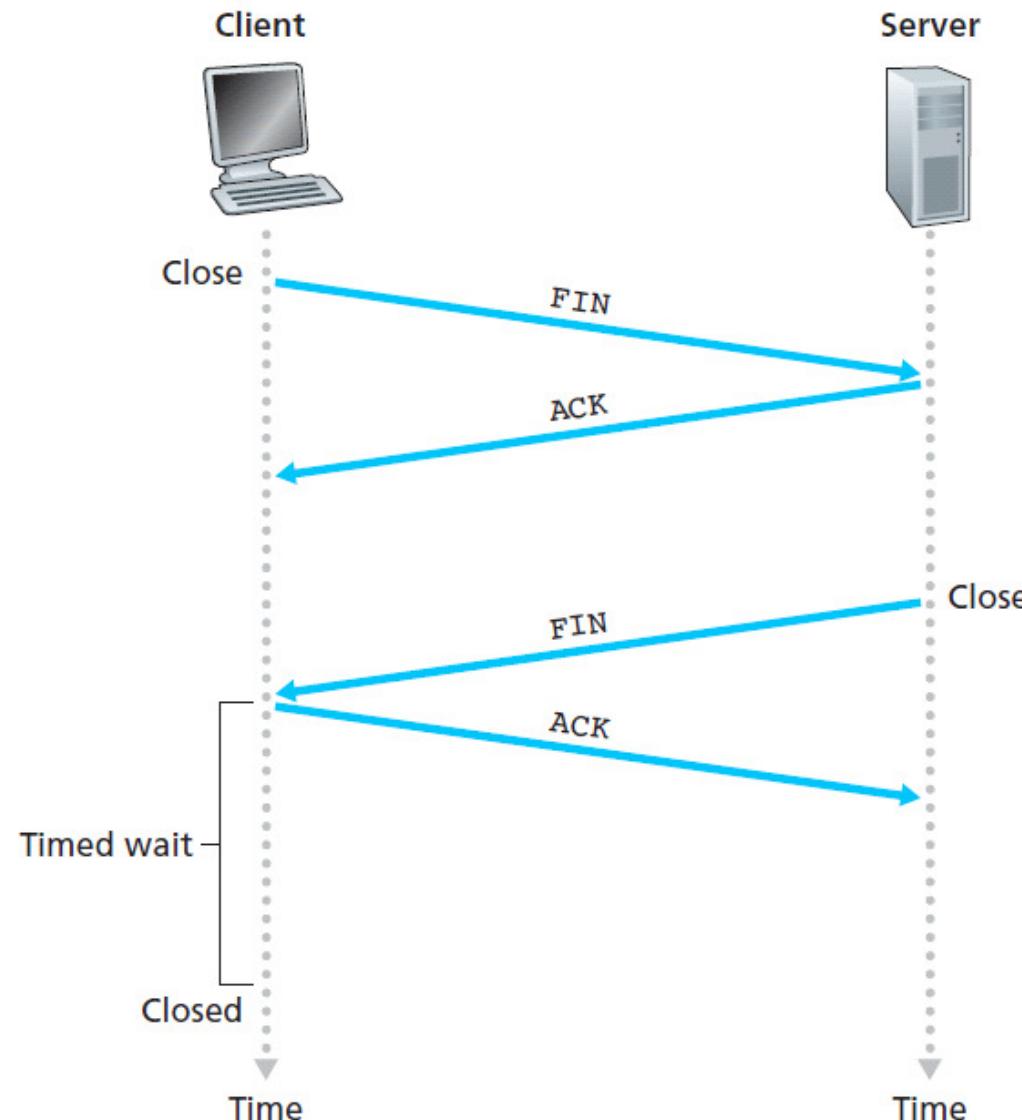
- **Semelhanças com o SR**

- Existe uma modificação proposta para o TCP denominada **reconhecimento seletivo**;
- Ela permite que um destinatário reconheça seletivamente segmentos fora de ordem, em vez de apenas reconhecer cumulativamente o último segmento recebido corretamente e na ordem.

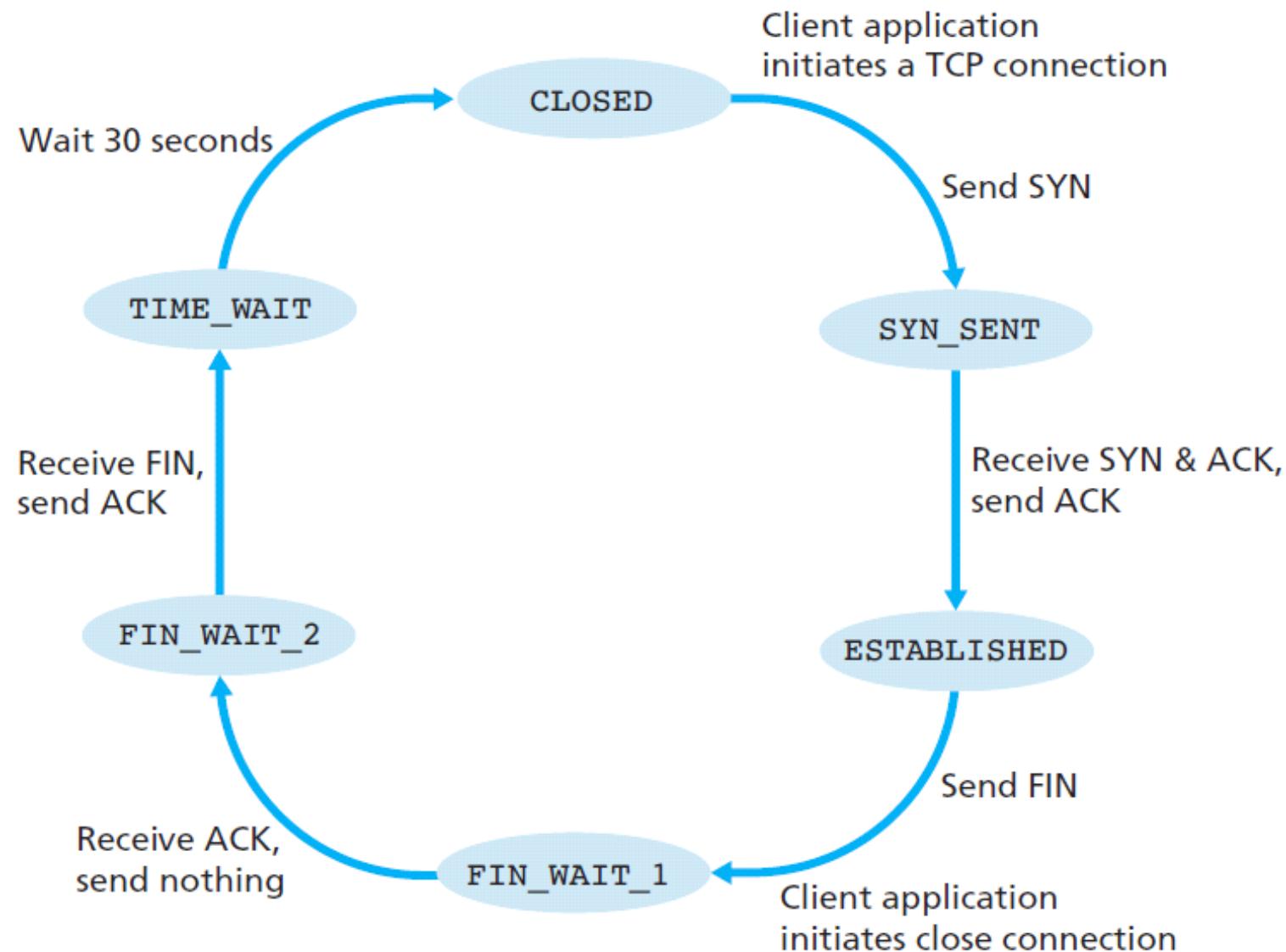
Apresentação de três vias (triplo handshake)



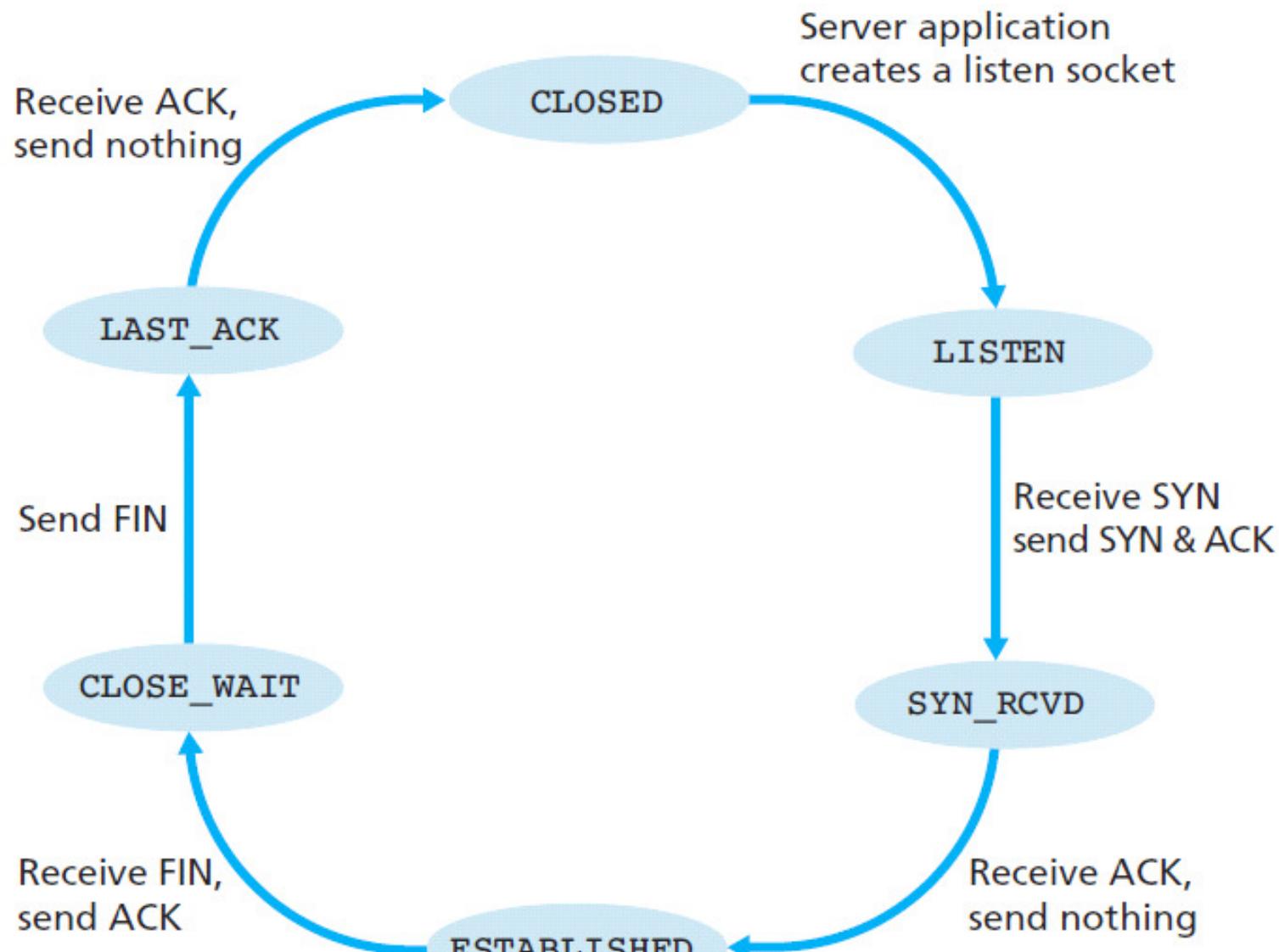
Encerramento de uma conexão TCP



Sequência típica de estados do TCP cliente



Sequência típica de estados do TCP servidor



Controle de Fluxo

- Se a aplicação for relativamente **lenta na leitura** dos dados, o remetente pode muito facilmente **saturar o buffer de recepção** da conexão por enviar demasiados dados muito rapidamente;

Controle de Fluxo

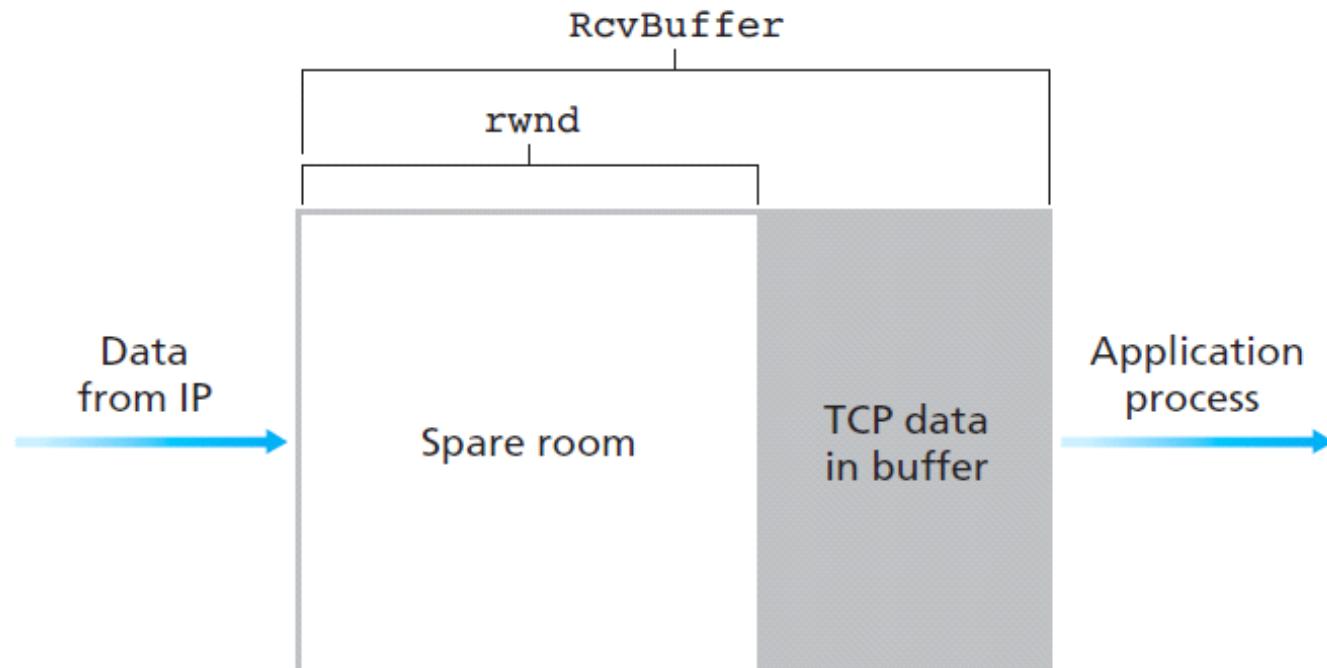
Serviço de compatibilização de velocidades, i.e., compatibiliza a taxa à qual o remetente está enviando com aquela à qual a aplicação receptora está lendo.

- O TCP remetente mantém uma variável denominada **janela de recepção**, a qual dá uma ideia do espaço de buffer livre disponível no destinatário.

Controle de Fluxo

- Como o TCP não tem permissão para saturar o buffer alocado no destinatário, devemos ter no host B:

$$LastByteRcvd - LastByteRead \leq RcvBuffer \quad (6)$$



Controle de Fluxo

- A janela de recepção é ajustada para a quantidade de espaço disponível no buffer, e depois reportada para o host A:

$$rwnd = RcvBuffer - (LastByteRcvd - LastByteRead) \quad (7)$$

- O host A tem certeza que não saturará o host B, certificando-se durante toda a conexão, de que:

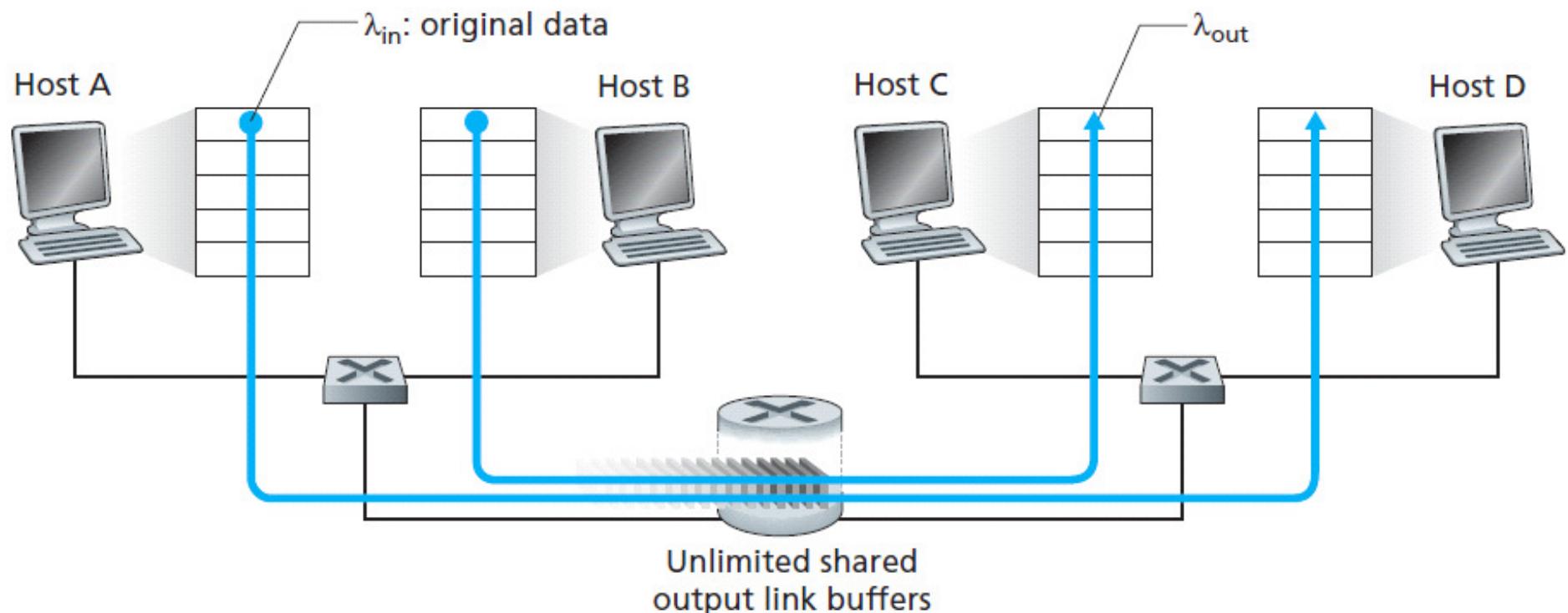
$$LastByteSent - LastByteAcked \leq rwnd \quad (8)$$

As causas e os custos do congestionamento

- O congestionamento é um dos “**dez mais**” da lista de problemas fundamentalmente importantes no trabalho em rede;
- A retransmissão de pacotes trata de um **sintoma** de congestionamento de rede (a perda de um segmento), mas não trata da causa do congestionamento da rede;
- A **causa** do congestionamento são demasiadas fontes tentando enviar dados a uma **taxa muito alta**.

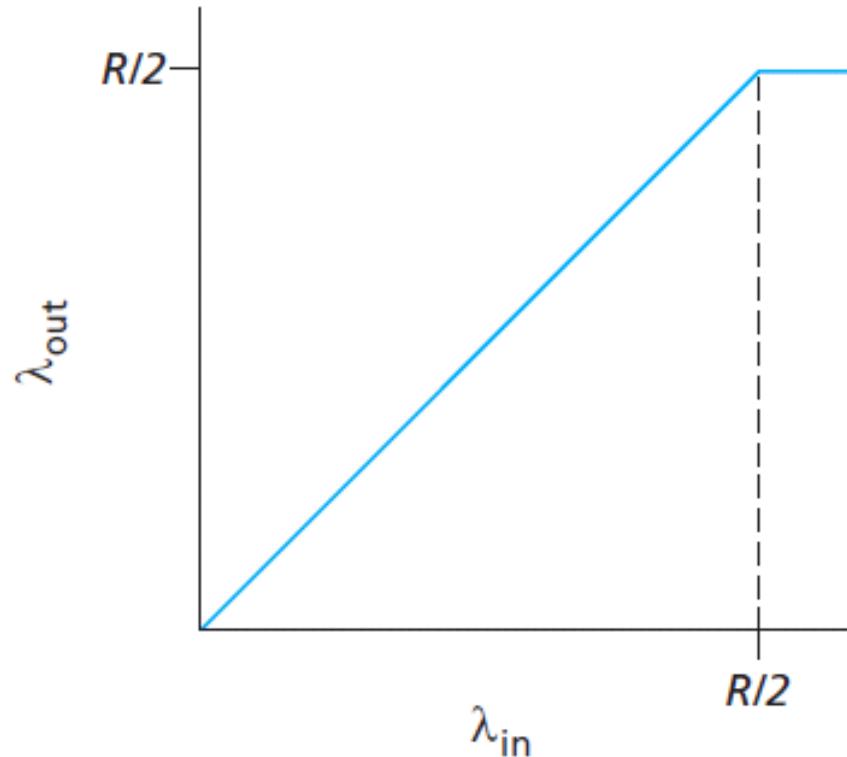
As causas e os custos do congestionamento

Cenário 1: dois remetentes, um roteador com buffers infinitos

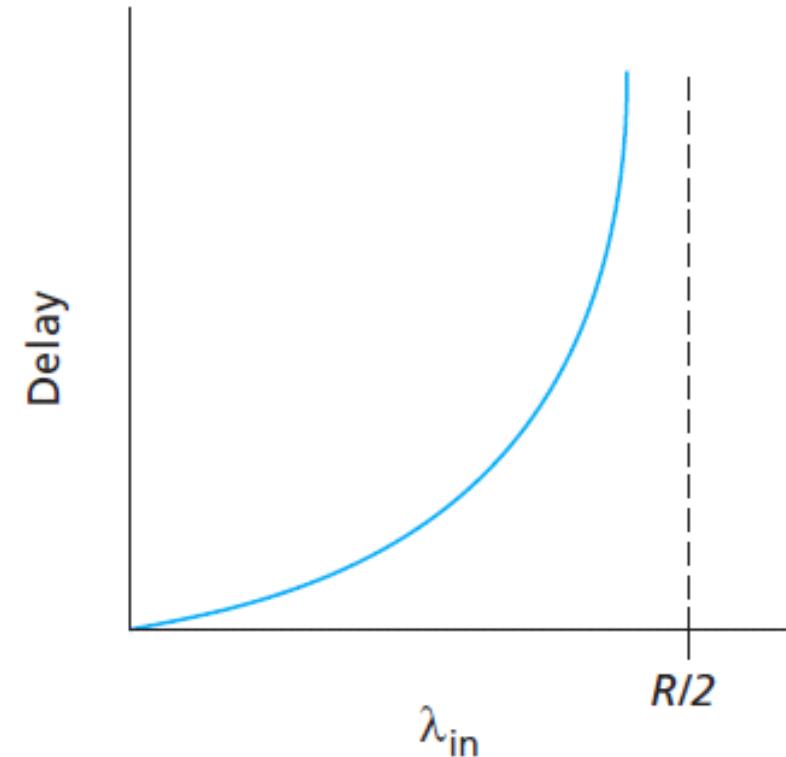


As causas e os custos do congestionamento

Cenário 1: dois remetentes, um roteador com buffers infinitos



a.



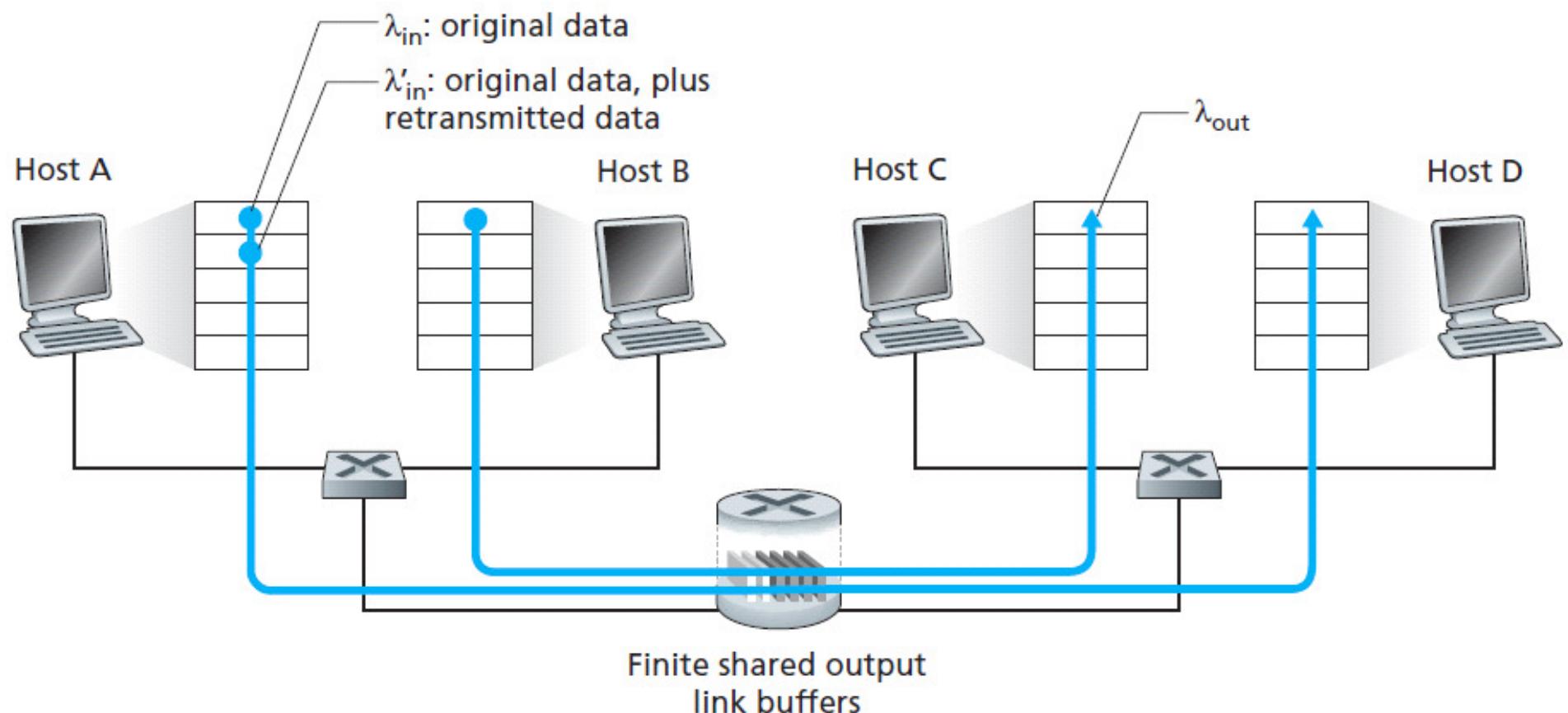
b.

Custo 1

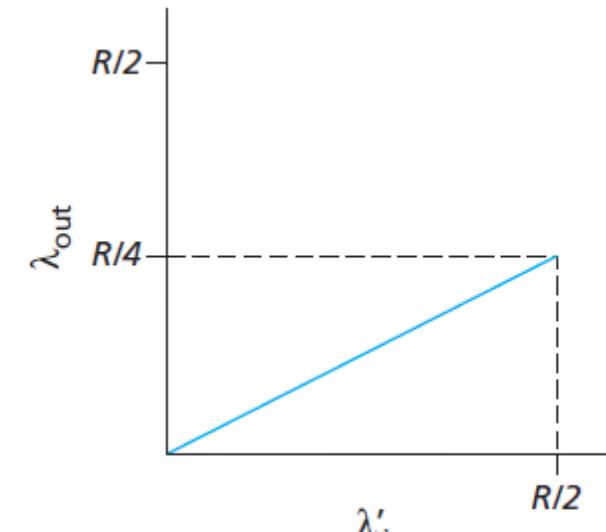
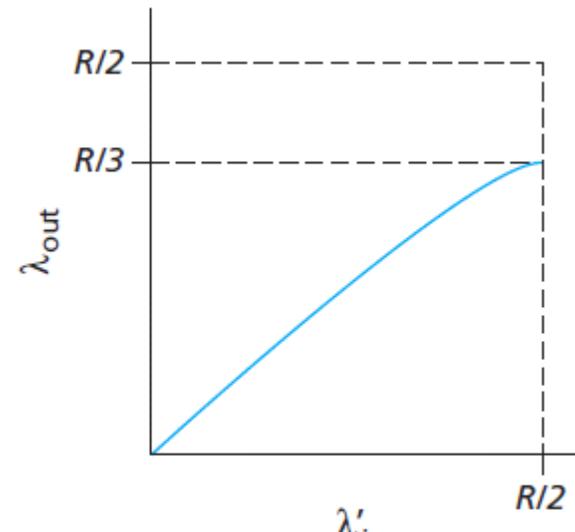
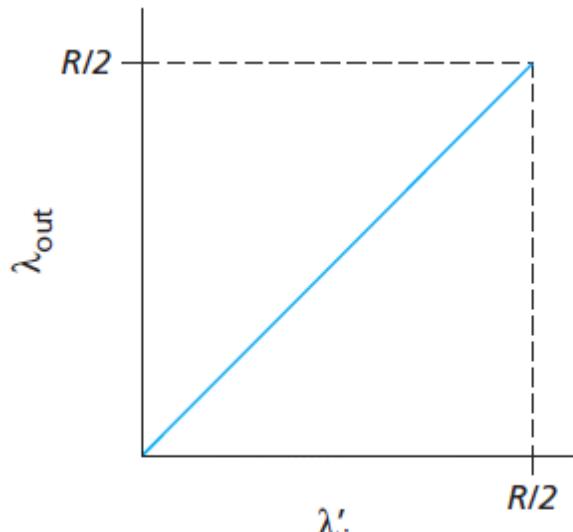
Há grandes atrasos de fila quando a taxa de chegada de pacotes se aproxima da capacidade do enlace.

As causas e os custos do congestionamento

Cenário 2: dois remetentes, um roteador com buffers finitos



As causas e os custos do congestionamento



Custo 2

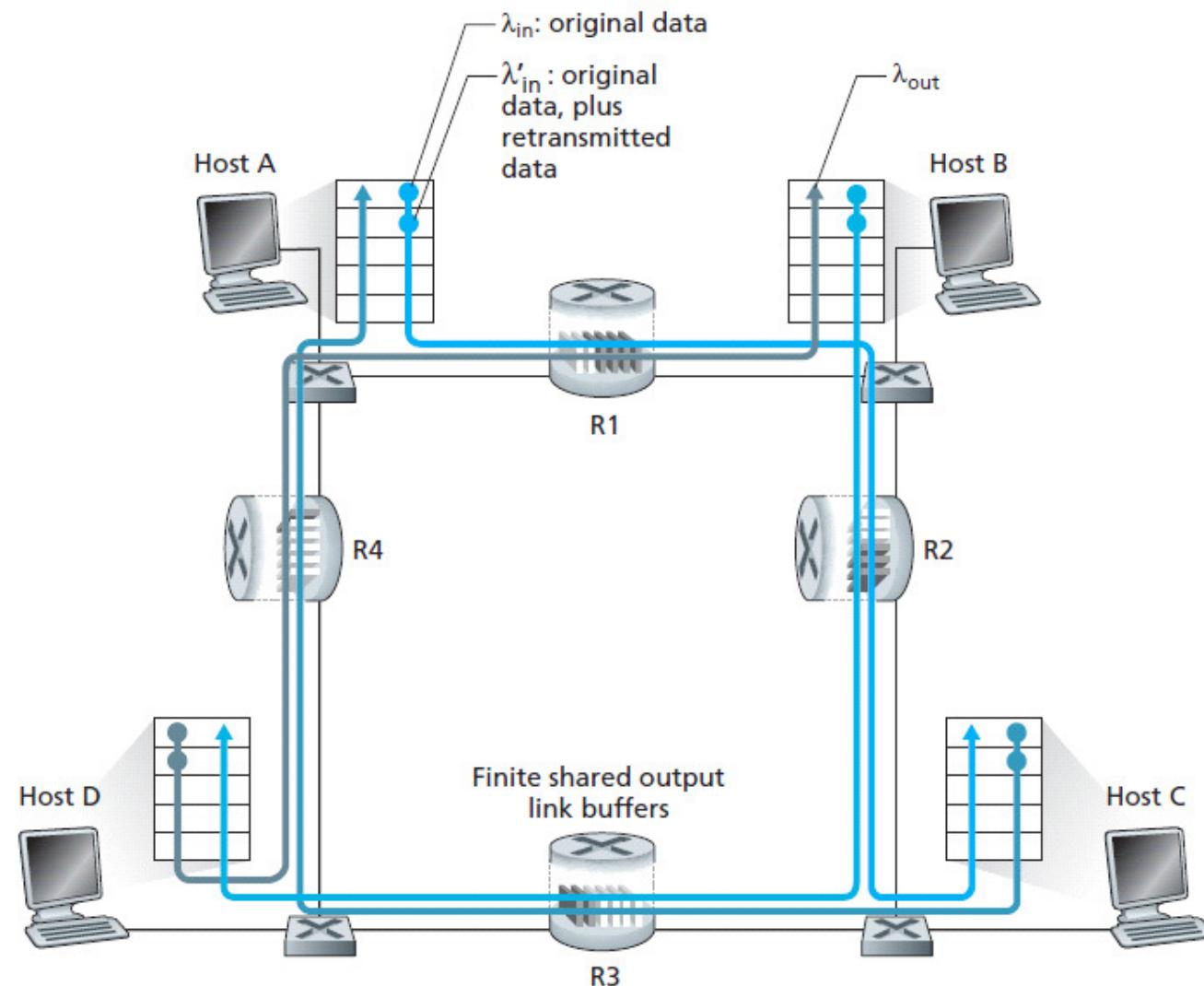
O remetente deve realizar transmissões para compensar os pacotes descartados (perdidos) devido à saturação do buffer.

Custo 3

Retransmissões desnecessárias feitas pelo remetente em face de grandes atrasos podem fazer com que um roteador use sua largura de banda de enlace para repassar cópias desnecessárias de um pacote.

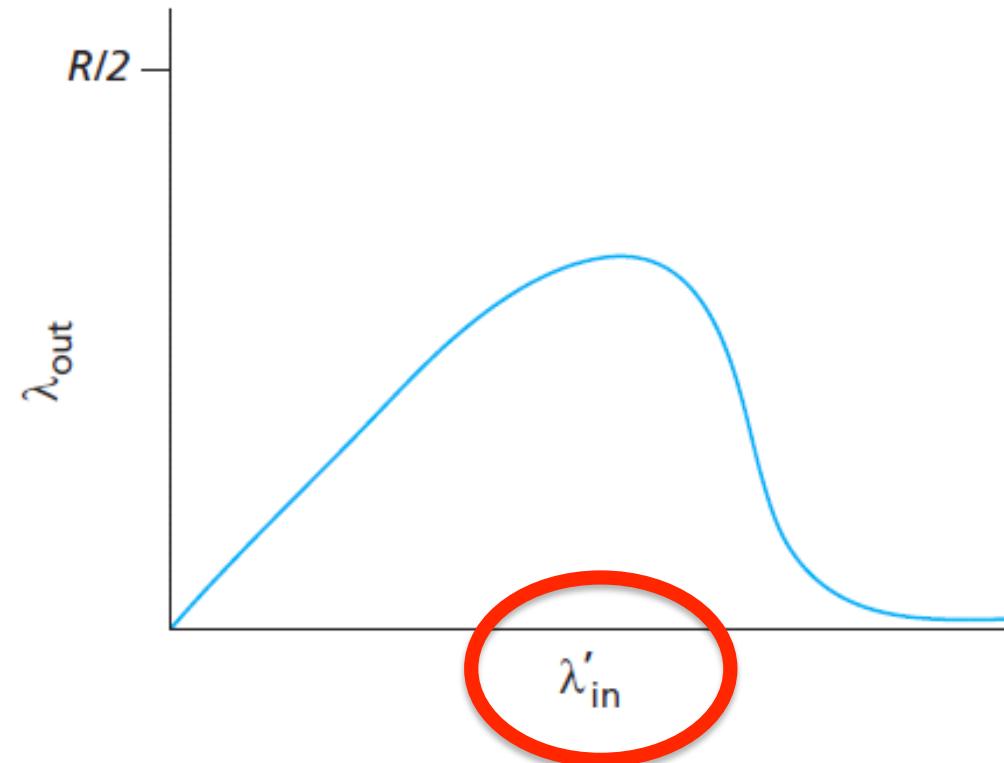
As causas e os custos do congestionamento

Cenário 3: quatro remetentes, roteadores com buffers finitos e trajetos com múltiplos roteadores



As causas e os custos do congestionamento

Cenário 3: quatro remetentes, roteadores com buffers finitos e trajetos com múltiplos roteadores



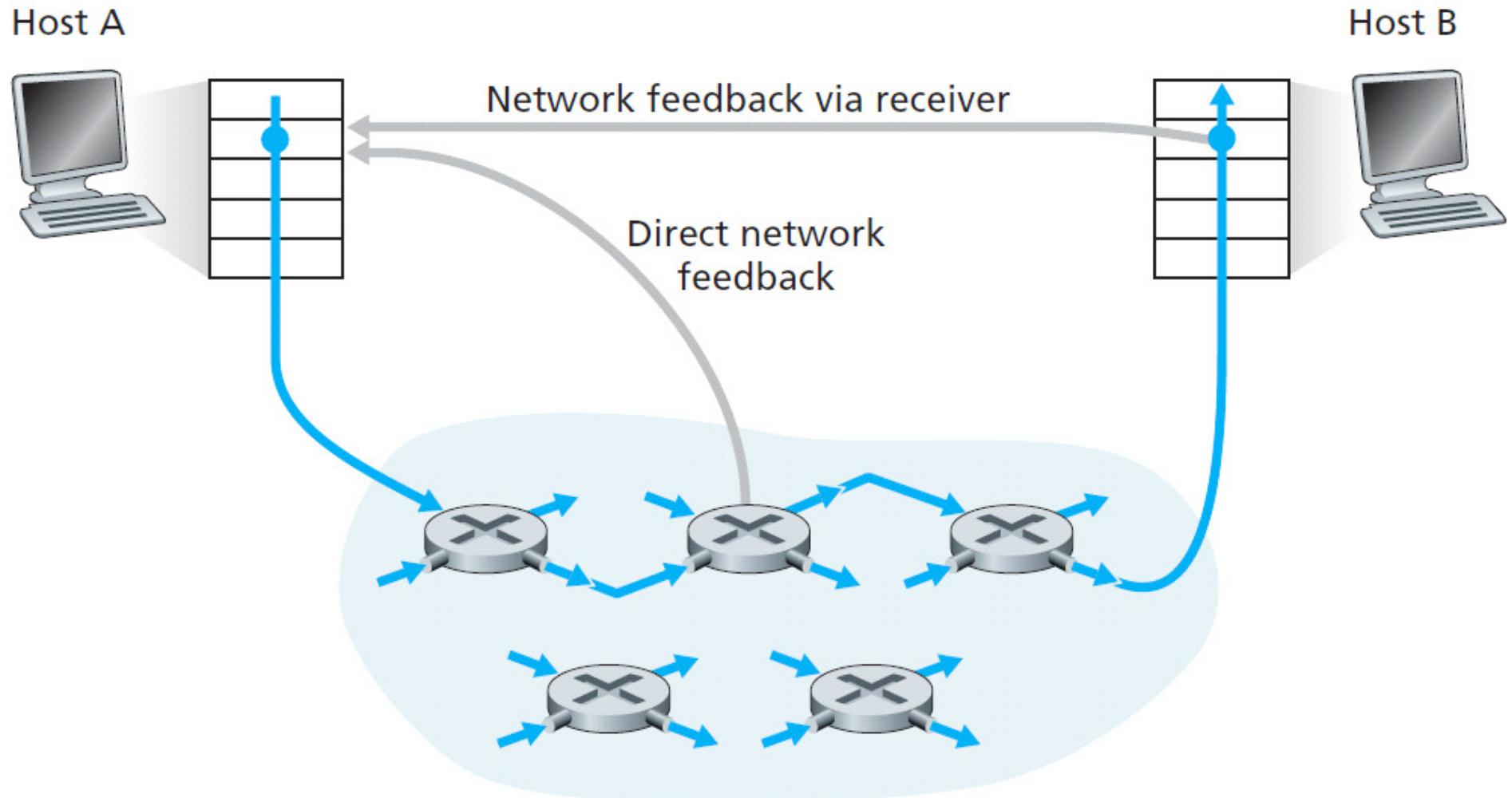
Custo 4

Quando um pacote é descartado ao longo do caminho devido a congestionamento, a capacidade de transmissão que foi usada em cada um dos enlaces anteriores para repassar o pacote até o ponto em que foi descartado acaba sendo desperdiçada.

Mecanismos de controle de congestionamento

- Dois procedimentos mais comuns adotados pelo controle de congestionamento:
 - **Controle de congestionamento fim a fim**
 - A presença de congestionamento na rede deve ser **intuída** pelos sistemas finais com base apenas na **observação** do comportamento da rede;
 - O TCP utiliza esse procedimento, considerando a **perda de um segmento** (timeout ou 3 ACKs duplicados) como indicação de congestionamento.
 - **Controle de congestionamento assistido pela rede**
 - Os roteadores da camada de rede fornecem **realimentação** específica de informações ao remetente a respeito do estado de congestionamento da rede;
 - Utilização de pacotes que indicam congestionamento: roteador remetente ou destinatário remetente.

Mecanismos de controle de congestionamento



Controle de congestionamento no TCP

- O TCP obriga cada remetente a **limitar a taxa** à qual enviam tráfego para sua conexão como **função do congestionamento** da rede percebido.
- O TCP é um protocolo **autorregulado**.

Como um remetente TCP limita a taxa à qual envia tráfego para sua conexão?

Controle de congestionamento no TCP

Como um remetente TCP limita a taxa à qual envia tráfego para sua conexão?

Utilizando uma janela de congestionamento `cwnd`.

$$LastByteSent - LastByteAcked \leq \min\{cwnd, rwnd\} \quad (9)$$

- A taxa de envio do remetente é aproximadamente $cwnd/RTT$ bytes por segundo. Portanto, ajustando o valor de $cwnd$, o remetente pode ajustar a taxa à qual envia dados para sua conexão.

Controle de congestionamento no TCP

Como um remetente TCP percebe que há congestionamento entre ele e o destinatário?

Controle de congestionamento no TCP

Como um remetente TCP percebe que há congestionamento entre ele e o destinatário?

Observando um “evento de perda”: timeout de temporização ou recebimento de 3 ACKs duplicados do destinatário.

Que algoritmo o remetente deve utilizar para modificar sua taxa de envio como uma função do congestionamento fim a fim?

Controle de congestionamento no TCP

Que algoritmo o remetente deve utilizar para modificar sua taxa de envio como uma função do congestionamento fim a fim?

- **Diminuição de taxa:** Um segmento perdido implica em congestionamento, portanto, a taxa do remetente TCP deve diminuir quando um **segmento é perdido**;
- **Aumento de taxa:** Um **segmento reconhecido** indica que a rede está enviando os segmentos do remetente ao destinatário e, por isso, a taxa do remetente pode aumentar quando um ACK chegar para um segmento anteriormente não reconhecido;
- **Busca por largura de banda:** o remetente TCP aumenta sua taxa de transmissão para buscar a taxa a qual o congestionamento se inicia, recua dessa taxa e novamente faz a busca para ver se a taxa de início do congestionamento foi alterada.

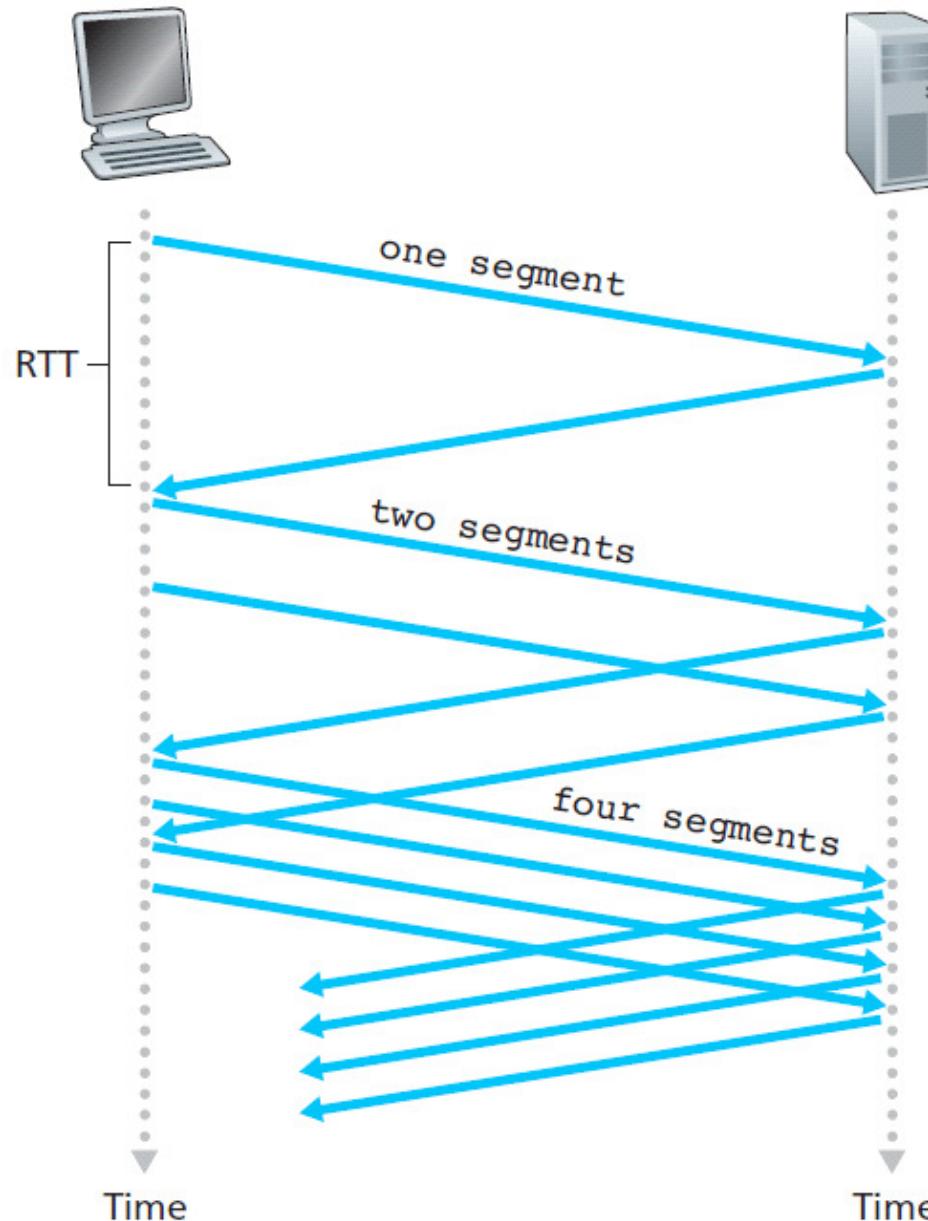
Controle de congestionamento no TCP

- O algoritmo do controle de congestionamento do TCP tem 3 componentes principais:
 - 1 **Partida lenta;**
 - 2 **Prevenção de congestionamento;**
 - 3 **Recuperação rápida.**

Partida lenta

- Quando uma conexão TCP começa, o valor de cwnd normalmente é **inicializado em 1 MSS**, resultando em uma taxa inicial de envio de MSS/RTT;
- O valor da janela de congestionamento **aumenta 1 MSS** toda vez que um segmento é reconhecido;
- Esse processo resulta em uma **multiplicação da taxa** de envio a cada RTT;
- A taxa de envio TCP se inicia lenta, mas **cresce exponencialmente** durante a fase de partida lenta.

Partida lenta



Partida lenta

Mas em que momento esse crescimento exponencial termina?

Se houver um evento de perda (timeout ou ACKs duplicados) ou se o limiar de partida lenta (**ssthresh**) for alcançado.

- **Evento de perda (timeout temporizador)**
 - O remetente TCP estabelece o valor de $cwnd$ para 1 e inicia o processo de partida lenta novamente (crescimento exponencial da janela);
 - Calcula o valor do limiar de partida lenta nesse momento do congestionamento: $ssthresh = cwnd/2$.
- **O limiar **ssthresh** é ultrapassado**
 - Pode ser uma atitude precipitada continuar duplicando $cwnd$ ao atingir ou ultrapassar o valor de $ssthresh$;
 - Quando isso ocorrer, a partida lenta termina e o TCP é alterado para o modo de **prevenção de congestionamento**.
- **Evento de perda (3 ACKs duplicados)**
 - O TCP entra no estado de **recuperação rápida**.

Prevenção de congestionamento

- Ao entrar no estado de prevenção de congestionamento, o valor de cwnd é aproximadamente metade de seu valor quando o congestionamento foi encontrado pela última vez;
- Em vez de duplicar o valor de cwnd a cada RTT, o TCP adota uma abordagem mais conservadora e aumenta o valor de cwnd por meio de um único MSS a cada RTT;
- **Aumento linear** da janela de congestionamento.

Prevenção de congestionamento

Mas em que momento esse crescimento linear deve terminar?

Se houver um evento de perda (timeout ou ACKs duplicados).

- **Evento de perda (timeout temporizador)**

- O remetente TCP estabelece o valor de $cwnd$ para 1 e inicia o processo de partida lenta novamente (crescimento exponencial da janela);
- Calcula o valor do limiar de partida lenta nesse momento do congestionamento: $ssthresh = cwnd/2$.

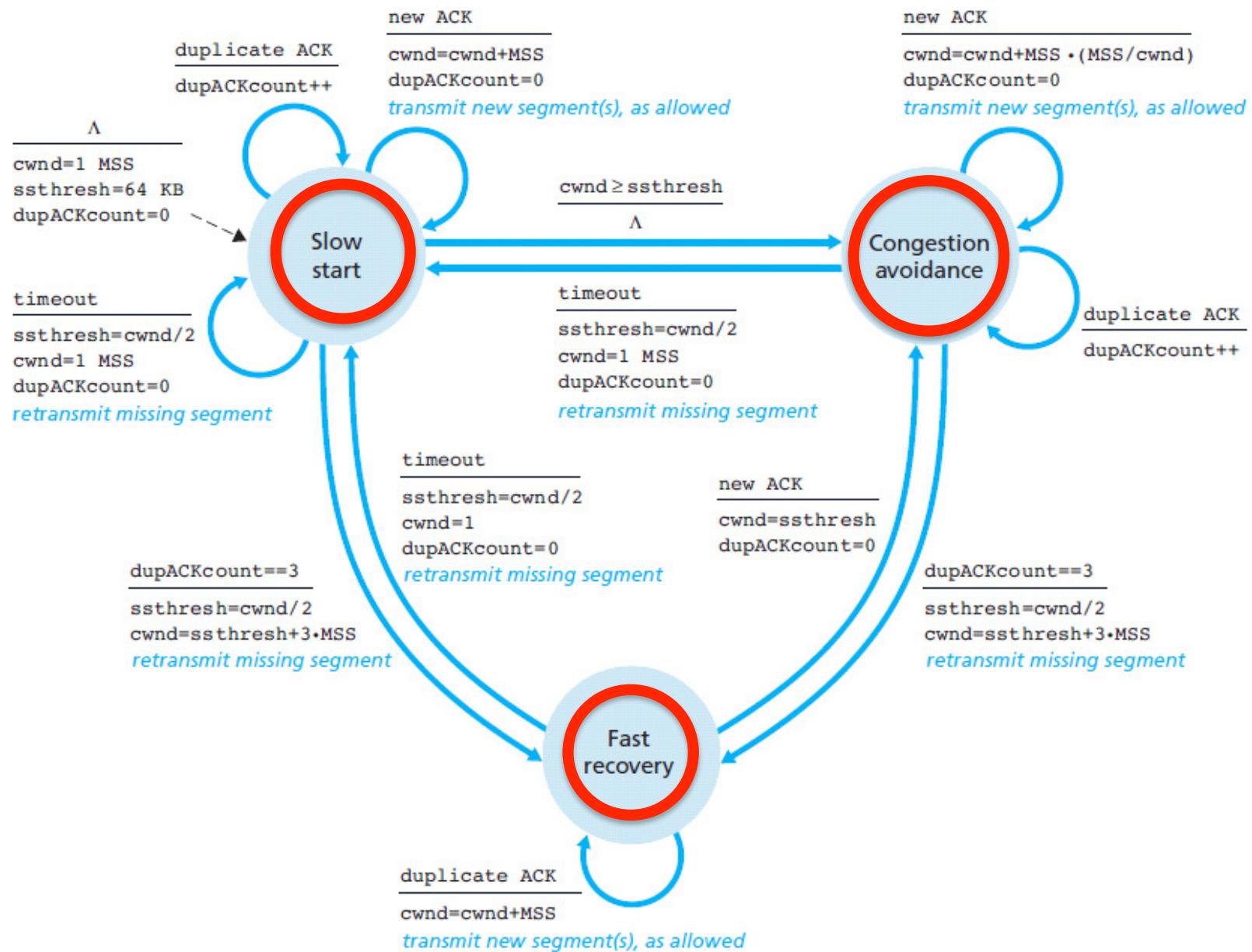
- **Evento de perda (3 ACKs duplicados)**

- Reação menos drástica: reduz o valor de $cwnd$ para metade (adicionando 3 MSSs) e registra o valor de $ssthresh$ como metade do $cwnd$ quando o evento dos ACKs duplicados ocorreu;
- O TCP entra no estado de **recuperação rápida**.

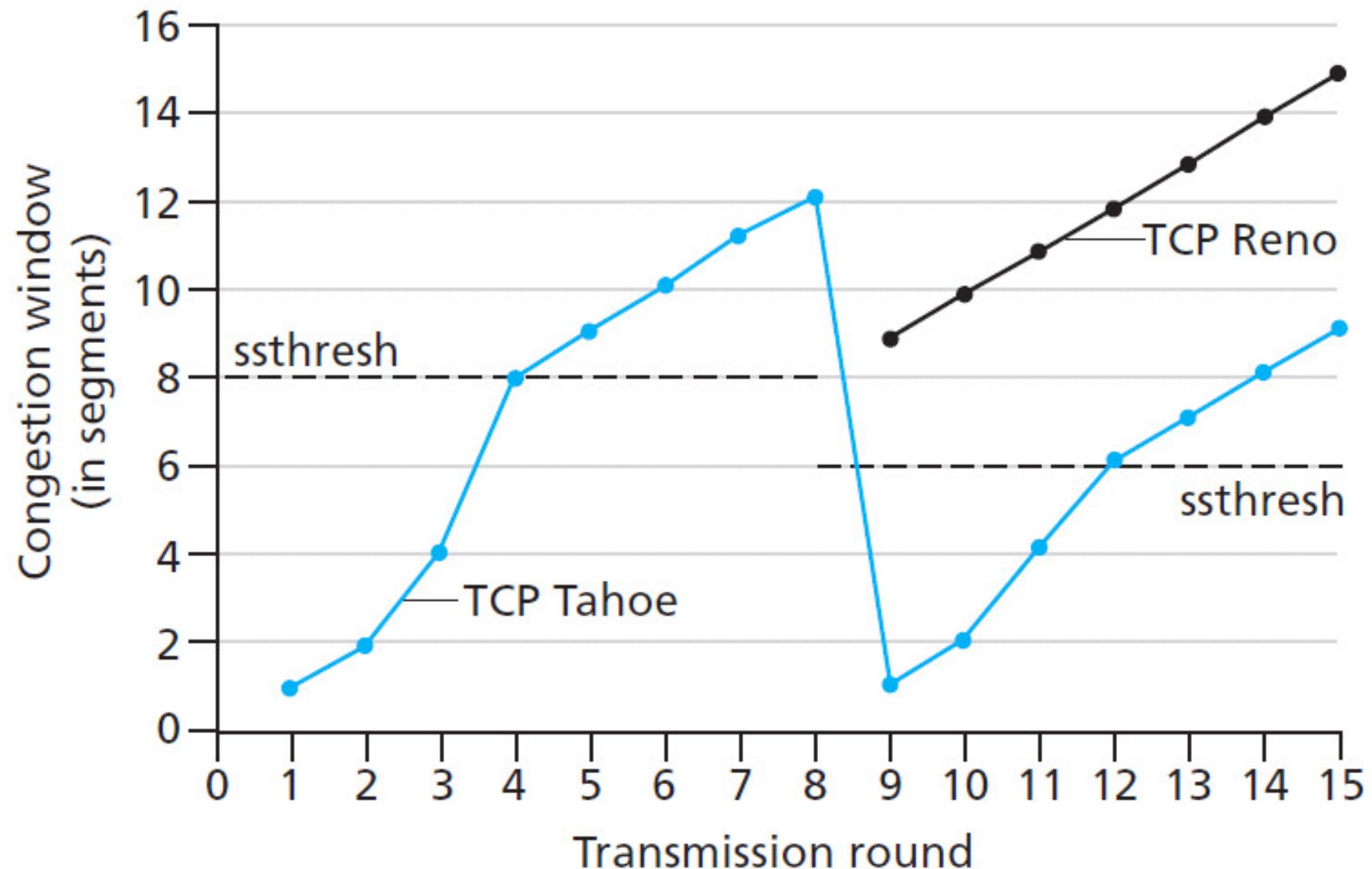
Recuperação rápida

- O valor de `cwnd` é aumentado por 1 MSS para cada ACK duplicado recebido no segmento perdido que fez com que o TCP entrasse no modo de recuperação rápida (aumento linear);
- Se um evento de timeout ocorrer, a recuperação rápida é alterada para o modo de partida lenta: o valor de `cwnd` é ajustado para 1 MSS, e o valor de `ssthresh`, para metade do valor de `cwnd` no momento em que o evento de perda ocorreu.

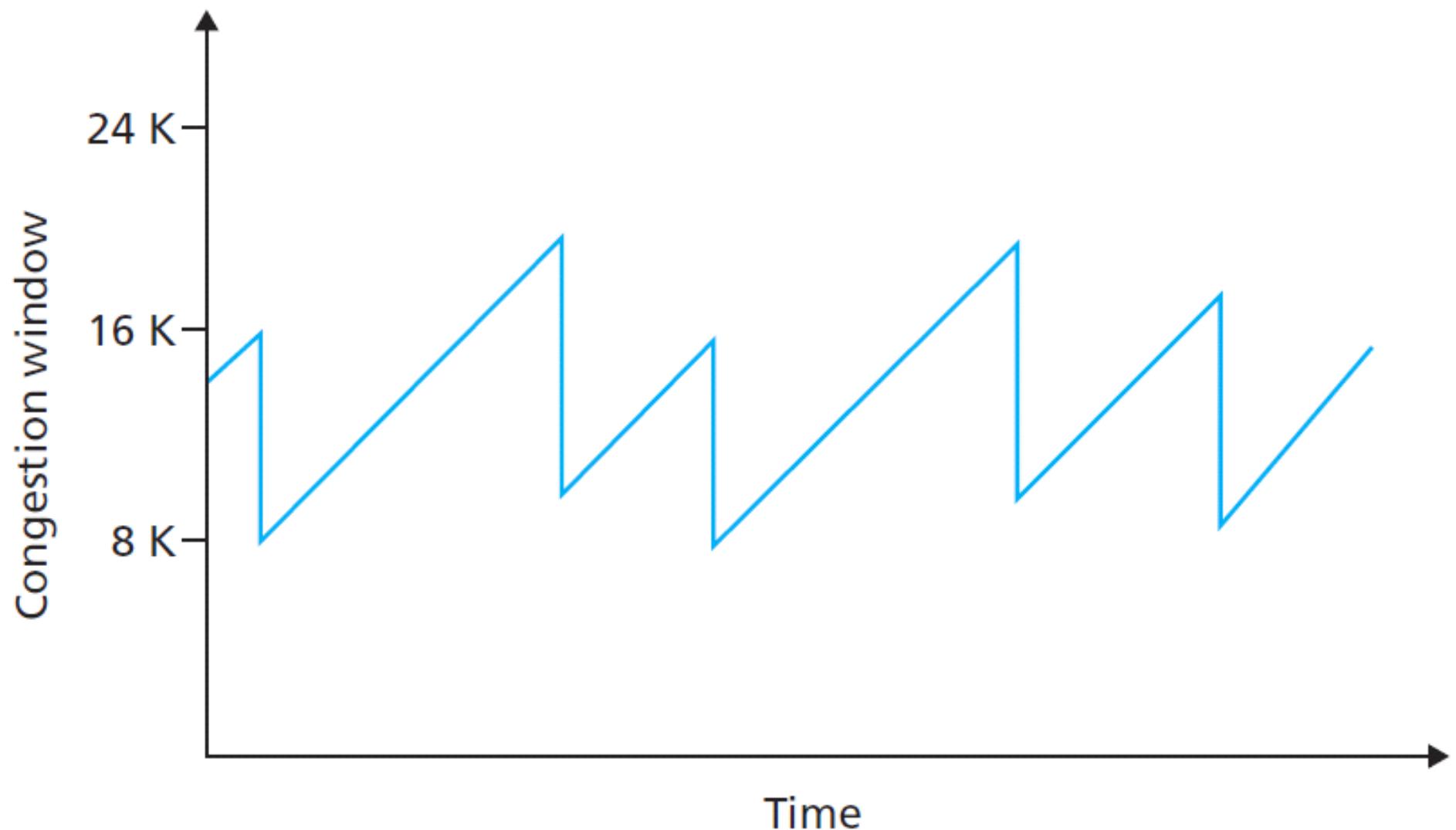
FSM do TCP



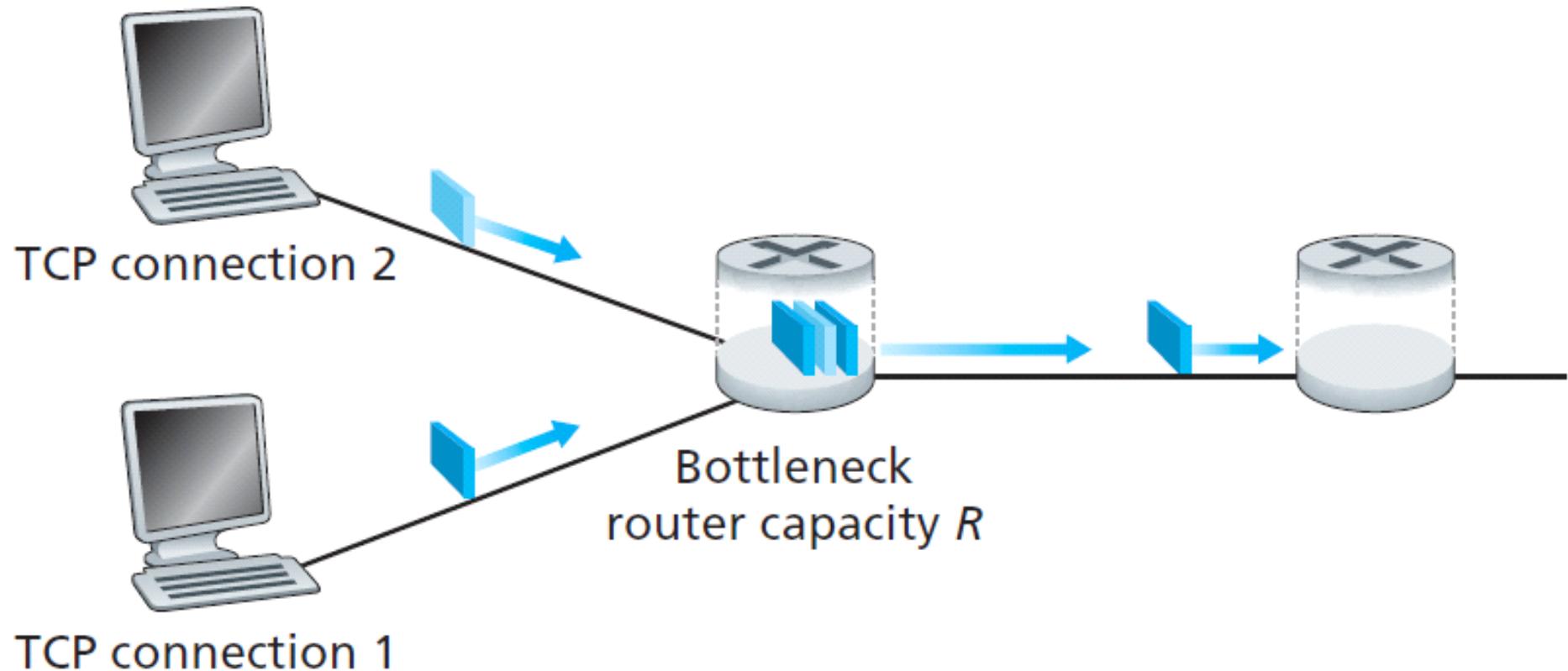
Evolução da janela de congestionamento do TCP



Descrição macroscópica da dinâmica do TCP



Equidade (Justiça)



Equidade (Justiça)

