UNIVERSIDADE FEDERAL DE PELOTAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

TEC2/TEC4: REDES MULTIMÍDIA (RMM)

Unidade 7
Revisão de Redes de Computadores
Camadas de Aplicação e Transporte

Prof. Guilherme Corrêa
gcorrea@inf.ufpel.edu.br

Sumário

- Introdução
- Camada de Aplicação
- Camada de Transporte

Sumário

- **Introdução**
- Camada de Aplicação
- Camada de Transporte

O que é um Protocolo?

Protocolos Humanos:

- "Que horas são?"
- "Eu tenho uma pergunta"
- Apresentações

… msgs específicas enviadas

… ações específicas tomadas quando msgs específicas são recebidas, ou outros eventos ocorrem

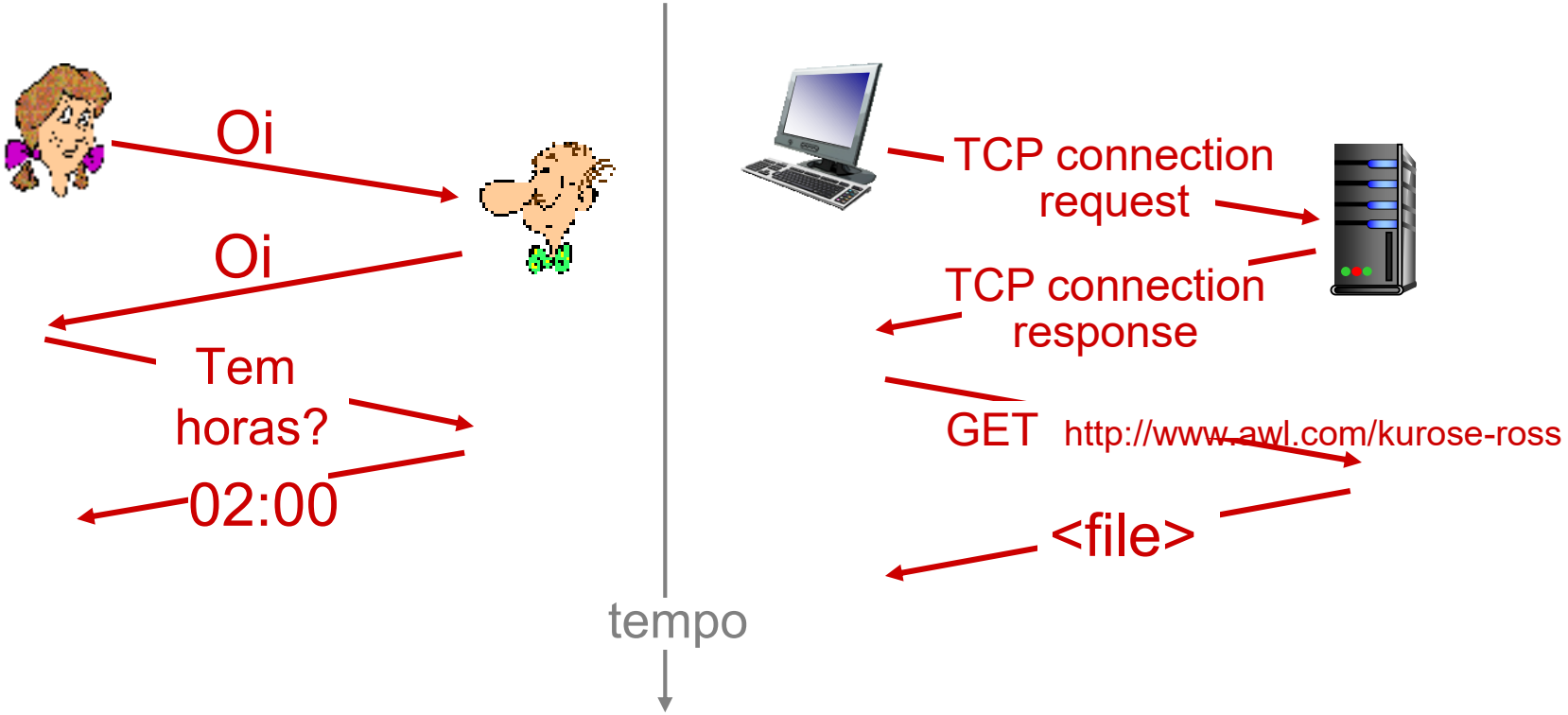
Protocolos de Redes:

- Máquinas ao invés de humanos
- Todas as atividades de comunicação na Internet são governadas por protocolos

Protocolos definem formato e ordem das msgs trocadas entre entidades na rede, bem como ações realizadas na transmissão e/ou recebimento de uma msg.

O que é um Protocolo?

Um protocolo humano e um protocolo de rede:



Arquitetura de Camadas

Redes são complexas, com muitos "pedaços" :

- hospedeiros
- roteadores
- enlaces de vários meios
- aplicações
- protocolos
- hardware, software

Pergunta:

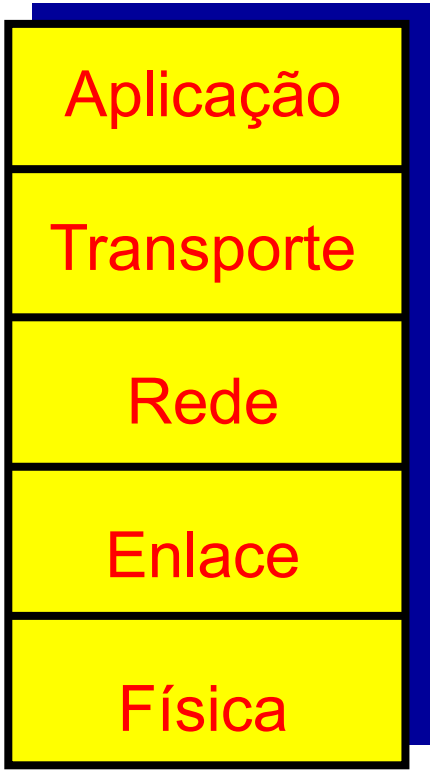
Há alguma esperança de organizar a estrutura da rede?

…. ou pelo menos o nosso estudo de redes?

Arquitetura de Camadas

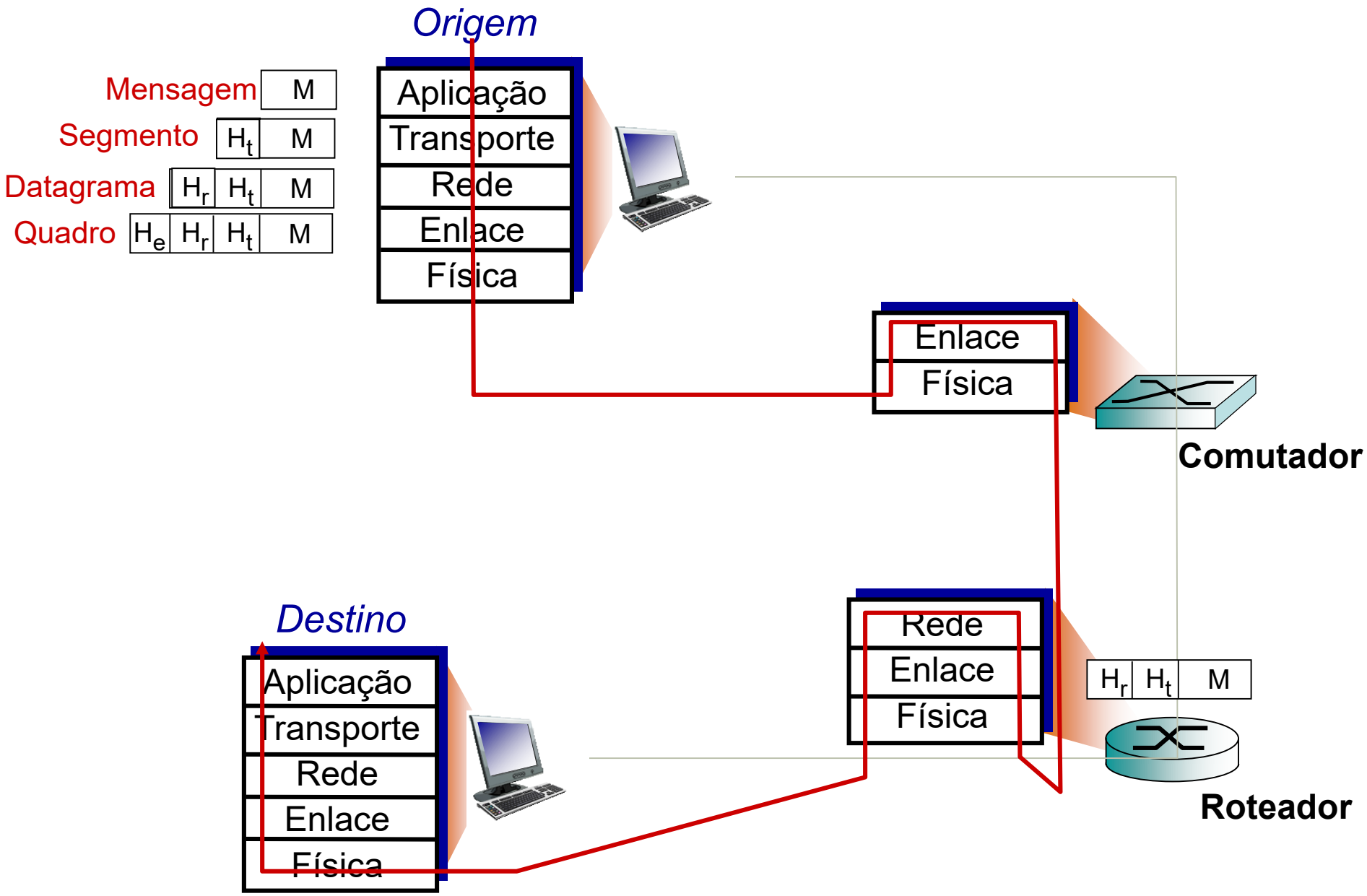
Pilha de Protocolos da Internet
(ou Modelo TCP/IP)

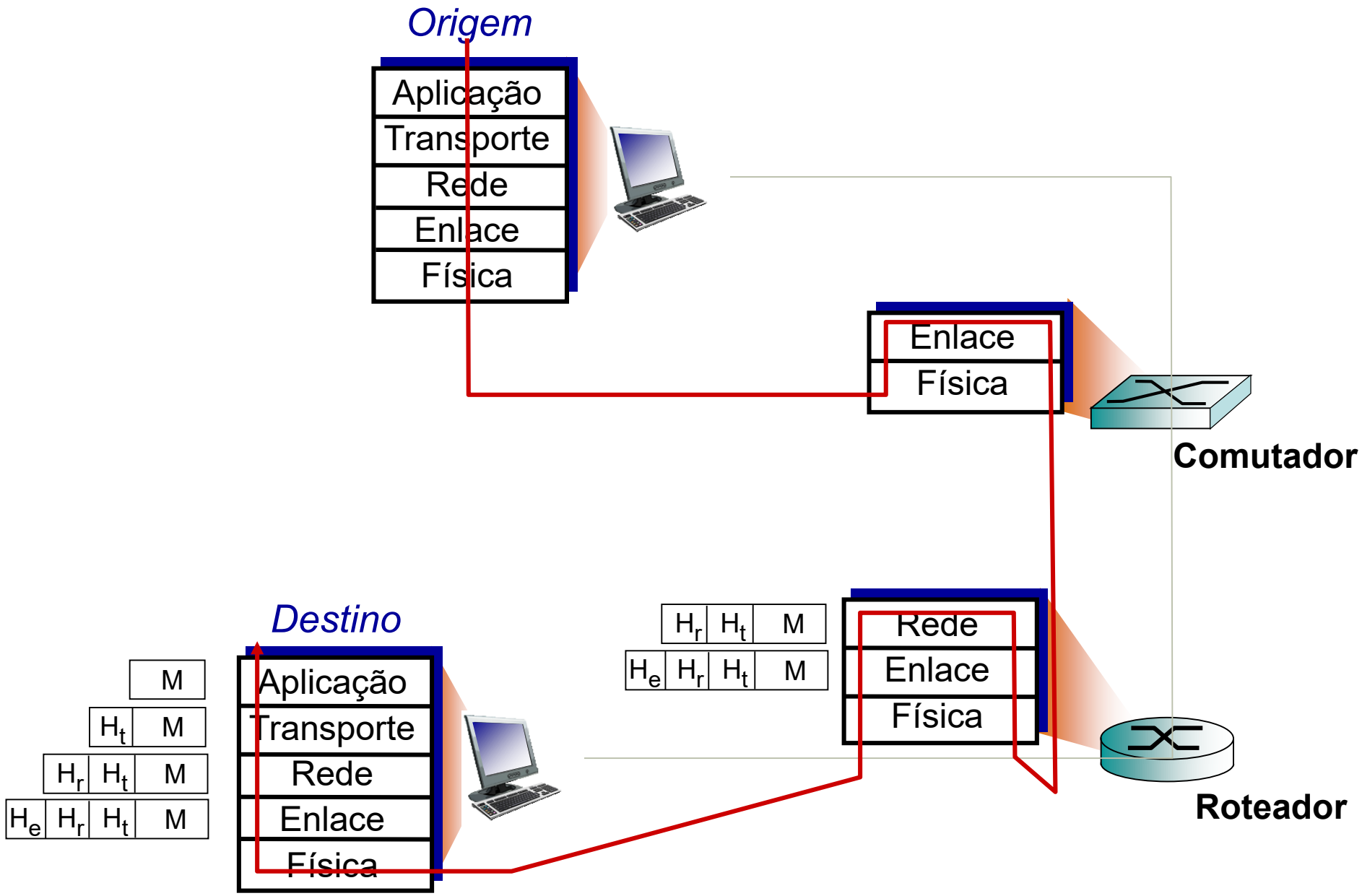
- *Aplicação:* dá suporte às aplicações de rede
  - FTP, SMTP, HTTP
- *Transporte:* transferência de dados processo-processo
  - TCP, UDP
- *Rede:* roteamento de datagramas de origem a destino
  - IP, protocolos de roteamento
- *Enlace:* transferência de dados entre elementos de rede vizinhos
  - Ethernet, 802.111 (WiFi), PPP
- *Física:* bits "na linha"



Arquitetura de Camadas

- Cada camada, combinada com as que estão abaixo dela, implementa alguma funcionalidade, algum **serviço**.
- Uma arquitetura de camadas nos permite discutir uma parcela específica e bem definida de um sistema grande e complexo.
- A modularidade facilita a atualização de componentes de sistema.


Origem
Mensagem
M
Segmento
H_t
M
Datagrama
H_r
H_t
M
Quadro
H_e
H_r
H_t
M
Aplicação
Transporte
Rede
Enlace
Física
Enlace
Física
Comutador
Destino
Aplicação
Transporte
Rede
Enlace
Física
Rede
Enlace
Física
H_r
H_t
M
Roteador


Origem
Aplicação
Transporte
Rede
Enlace
Física
Enlace
Física
Comutador
Destino
M
Ht M
Hr Ht M
He Hr Ht M
Aplicação
Transporte
Rede
Enlace
Física
Hr Ht M
He Hr Ht M
Rede
Enlace
Física
Roteador

Sumário

- Introdução
- **Camada de Aplicação**
- Camada de Transporte

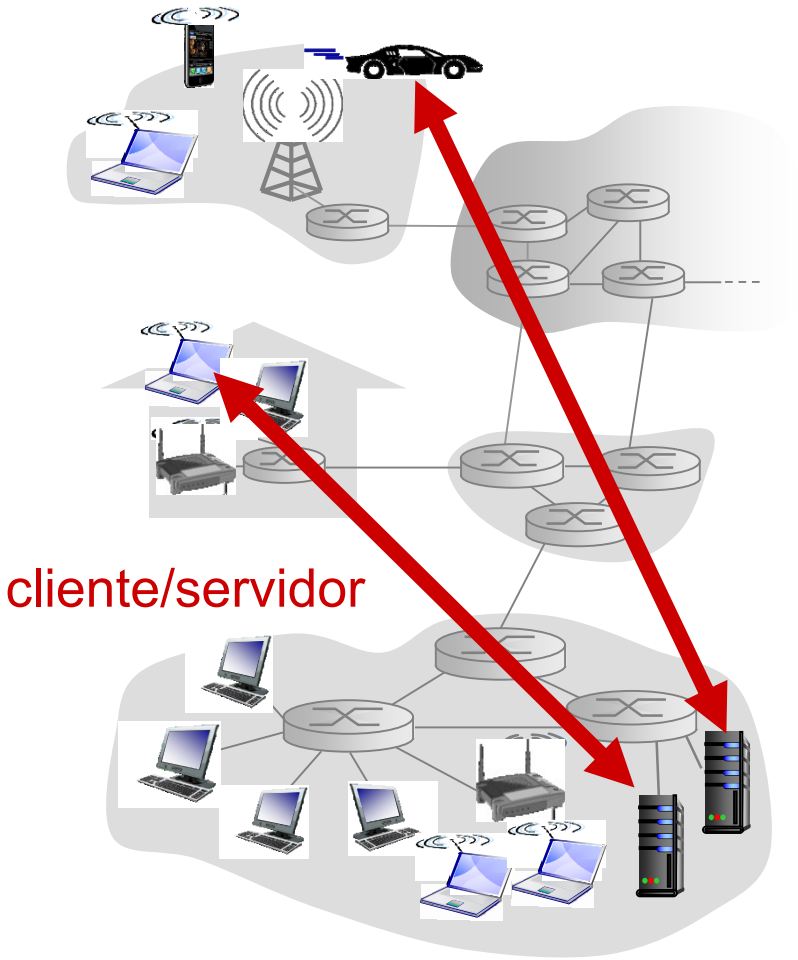
Camada de Aplicação: Introdução

Arquiteturas de aplicações de rede

- cliente-servidor
- *peer-to-peer* (P2P)

Camada de Aplicação: Introdução

Arquitetura Cliente-Servidor



servidor:

- hospedeiro "*always-on*"
- endereços IP permanentes
- data centers para escalonamento

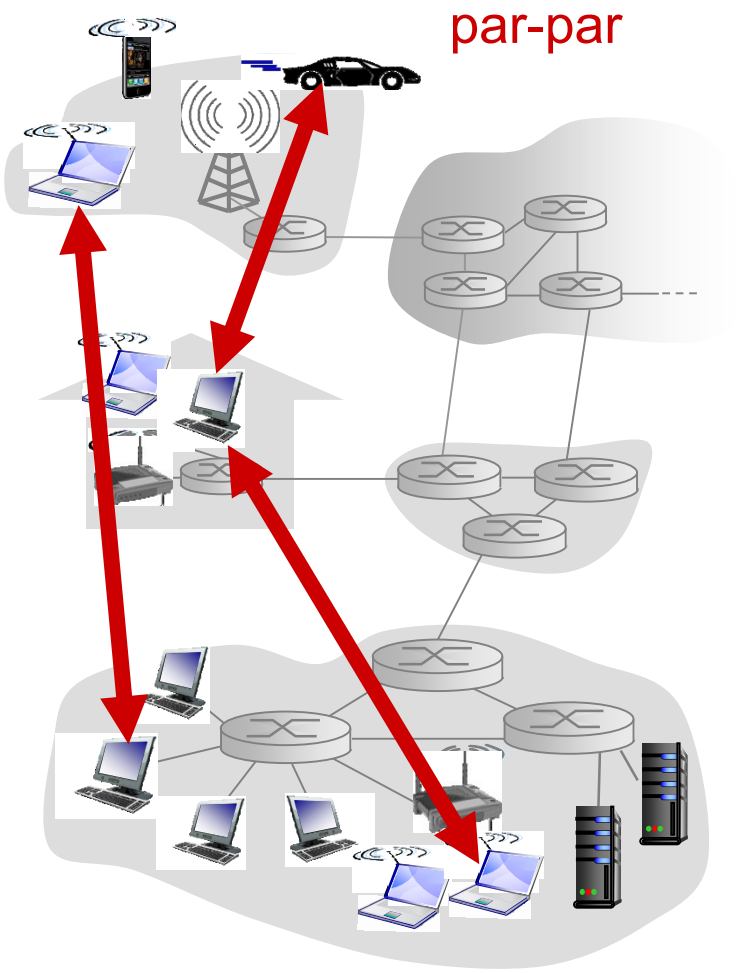
clientes:

- comunicam-se com servidor
- podem ser conectados intermitentemente
- podem ter endereços de IP dinâmicos
- não se comunicam diretamente

Camada de Aplicação: Introdução

Arquitetura *Peer-to-Peer* (P2P)

- servidor **não** "*always-on*"
- comunicação direta entre duplas de hospedeiros, denominados **pares** (*peer*)
- um par requisita um serviço de outro par e fornece serviços a outros pares
  - *auto-escalabilidade* – novos pares trazem mais capacidade de serviço, assim como mais demanda
- pares são intermitentemente conectados e mudam de endereço IP
  - gerenciamento complexo



Camada de Aplicação: Introdução

Comunicação de Processos

processo: programa executado num hospedeiro

- no mesmo hospedeiro, dois processos comunicam-se usando comunicação inter-processos (definida pelo SO)
- processos em diferentes hospedeiros comunicam-se trocando mensagens

clientes, servidores

processo cliente: processo que inicia a comunicação

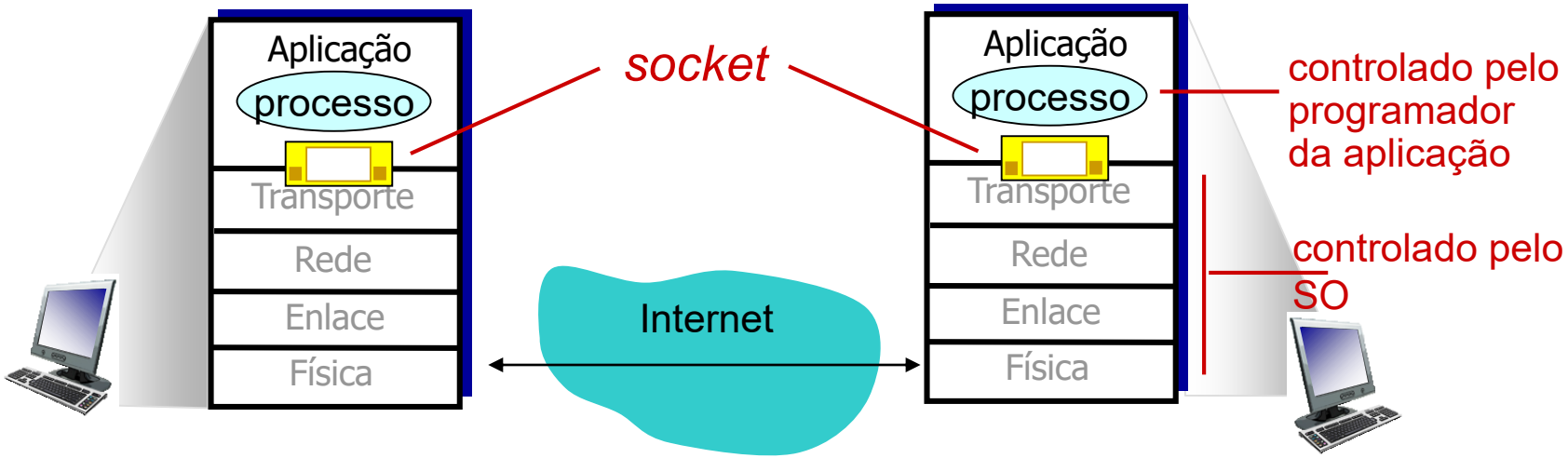
processo servidor: processo que espera para ser contatado

- aplicações com arquitetura P2P têm processos cliente & processos servidor

Camada de Aplicação: Introdução

Comunicação de Processos

- processo envia/recebe mensagens para/do seu ***socket***
- ***socket*** análogo a uma porta
  - o processo que envia "joga" a mensagem pela porta
  - o processo que envia admite que há uma infraestrutura de transporte no outro lado da porta para entregar a mensagem no *socket* do processo receptor



Comunicação de Processos

- para receber mensagens, os processos devem ter um *identificador*
- dispositivo hospedeiro tem um endereço de IP único de 32 bits
- *Pergunta:* o endereço de IP do hospedeiro em que o processo executa é suficiente para identificar o processo?

- *identificador* inclui endereço IP e número de porta associada ao processo no hospedeiro
- exemplos de núm. de porta:
  - servidor HTTP: 80
  - servidor de e-mail: 25
- para enviar mensagem HTTP ao servidor web gaia.cs.umass.edu:
  - endereço IP: 128.119.245.12
  - número de porta: 80

Camada de Aplicação: Introdução

Protocolos definem:

- tipos de mensagens trocadas,
  - ex.: requisição, resposta
- Sintaxe da mensagem:
  - quais campos e como estes são determinados
- Semântica da mensagem:
  - o significado dos campos das mensagens
- regras para quando e como os processo enviam e respondem as mensagens

protocolos abertos:

- definidos em RFCs
- permitem interoperabilidade
- ex.: HTTP, SMTP

protocolos proprietários:

- ex.: Skype

Camada de Aplicação: Introdução

Que serviços de transporte uma app precisa?

Algumas apps comuns:

Aplicação	Perda de dados	Vazão	Sensível ao tempo
transfer. de arquivos	sem perda	elástica	não
e-mail	sem perda	elástica	não
docs Web	sem perda	elástica	não
áudio/vídeo em tempo real	**tolera perda**	áudio: 5kbps-1Mbps vídeo:10kbps-5Mbps	**sim, 100's ms**
áudio/vídeo armazenado	**tolera perda**	igual acima	**sim, alguns segundos**
jogos interativos	tolera perda	poucos kbps-10kbps	sim, 100's ms
mensagem instantânea	sem perda	elástica	sim e não

Camada de Aplicação: Introdução

Serviços de transporte providos pela Internet

Serviços do TCP:

- *transporte confiável* entre processos comunicantes
- *controle de fluxo:* remetente não sobrecarrega receptor
- *controle de congestionamento:* limita o remetente quando a rede está sobrecarregada
- *não fornece:* temporização, vazão mínima, segurança
- *orientado a conexão:* processos cliente e servidor devem "se apresentar" e criar uma conexão antes de se comunicar

Serviços do UDP:

- *transferência de dados não confiável* entre processos comunicantes
- *não fornece:* confiabilidade, controle de fluxo, controle de congestionamento, temporização, garantia de vazão mínima, segurança, conexão

P: Mas então pra que serve o protocolo UDP?

Camada de Aplicação: Introdução

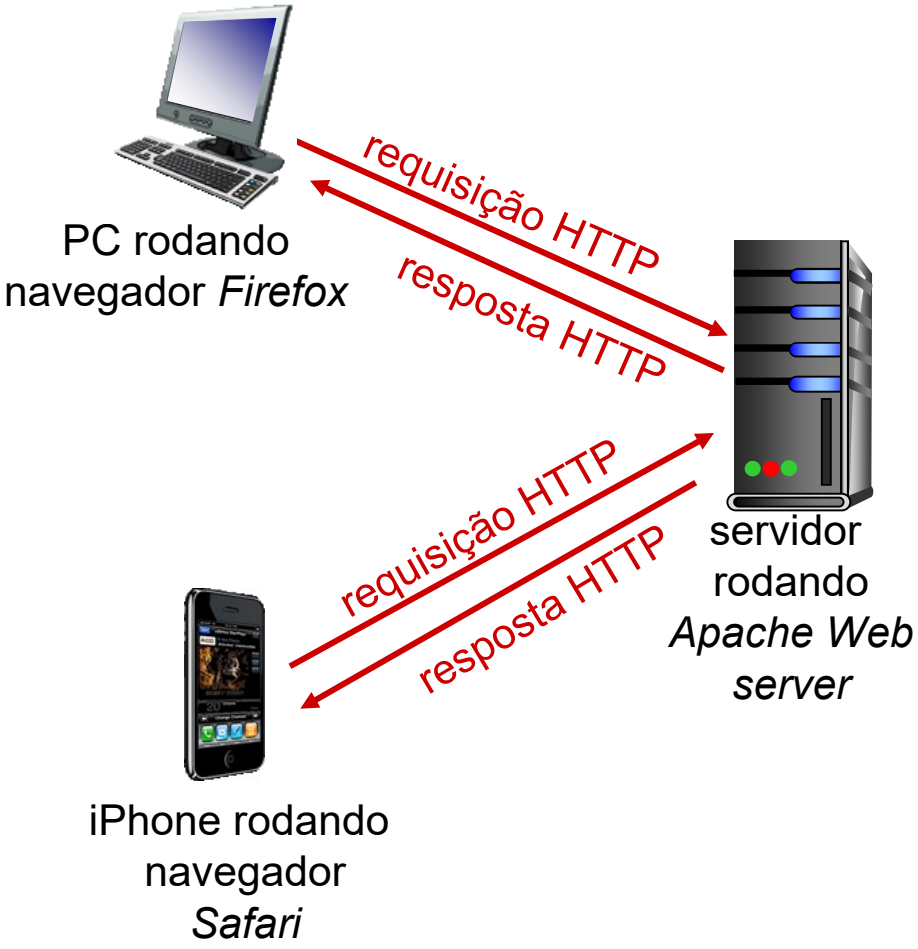
Protocolos de Aplicação e de Transporte

Aplicação	Protocolo na camada de Aplicação	Protocolo na camada de Transporte
e-mail	SMTP [RFC 2821]	TCP
acesso a terminal remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transferência de arquivo	FTP [RFC 959]	TCP
***streaming* multimídia**	**HTTP (e.g., YouTube), RTP [RFC 1889]**	**TCP ou UDP**
telefonia por Internet	**SIP, RTP, proprietário (e.g., Skype)**	**TCP ou UDP**

HTTP: Visão Geral

HTTP: hypertext transfer protocol

- protocolo da Web na camada de Aplicação
- modelo cliente/servidor
  - *cliente:* navegador requisita, recebe, (usando o protocolo HTTP) e "mostra" objetos Web
  - *servidor:* servidor Web envia (usando o protocolo HTTP) objetos em resposta às requisições



HTTP: Visão Geral

usa TCP:

- cliente inicia conexão TCP (cria *socket*) para o servidor, porta 80
- servidor aceita conexão TCP do cliente
- msgs HTTP (msgs de protocolo na camada de Aplicação) são trocadas entre navegador (cliente HTTP) e servidor Web (servidor HTTP)
- conexão TCP é fechada

*HTTP é "**sem estado**"*

- servidor não mantém informação sobre requisições passadas do cliente

Pq?

protocolos que mantêm "estado" são complexos

- histórico (estado) deve ser mantido
- se servidor/cliente falhar, as suas visões de "estado" podem ficar inconsistentes e precisam ser reconciliadas

HTTP não-persistente

suponha que o usuário entre com a URL:
`www.someSchool.edu/someDepartment/home.index`

(contém texto e referência a 10 imagens jpeg)

1a. cliente HTTP inicia conexão TCP ao servidor HTTP (processo) em www.someSchool.edu na porta 80

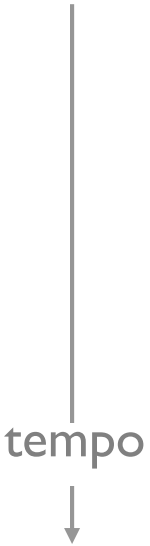
1b. servidor HTTP no hospedeiro www.someSchool.edu esperando por conexão TCP na porta 80. "aceita" conexão e notifica cliente

2. cliente HTTP envia *mensagem de requisição* HTTP (contendo URL) por meio do seu *socket* TCP. A mensagem indica que o cliente quer o objeto someDepartment/home.index

3. servidor HTTP recebe a mensagem de requisição, forma a *mensagem de resposta* contendo o objeto requisitado e envia a mensagem através do seu *socket*

tempo

HTTP não-persistente



4. servidor HTTP fecha a conexão TCP.

5. cliente HTTP recebe mensagem de resposta contendo o arquivo html e mostra o html. Ao examinar o arquivo html, encontra10 objetos jpeg referenciados

6. passos 1-5 repetidos para cada um dos 10 objetos jpeg

HTTP não-persistente

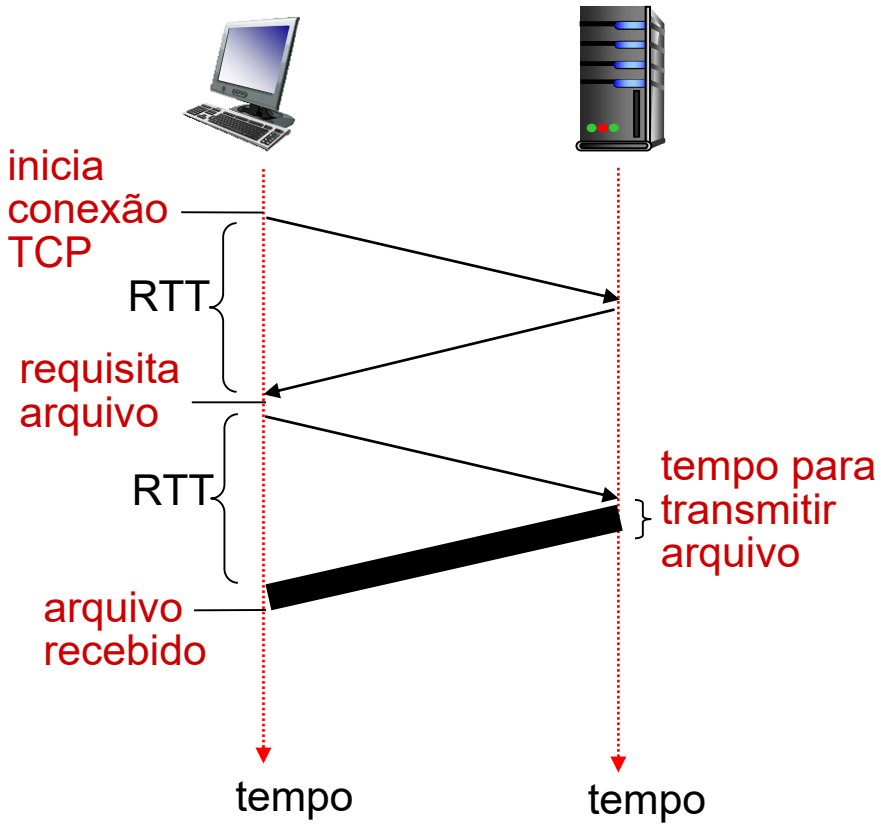
Tempo de resposta

RTT (*round-trip time*): "tempo de viagem de ida e volta" é o tempo para um pequeno pacote viajar do cliente ao servidor e de volta ao cliente

tempo de resposta HTTP:

- um RTT para iniciar conexão TCP
- um RTT para envio da requisição HTTP e recepção dos primeiros bytes da resposta HTTP
- tempo de transmissão do arquivo
- tempo de resposta do HTTP não-persistente =

  2RTT+ tempo de transmissão do arquivo



HTTP persistente

problemas no HTTP não-persistente:

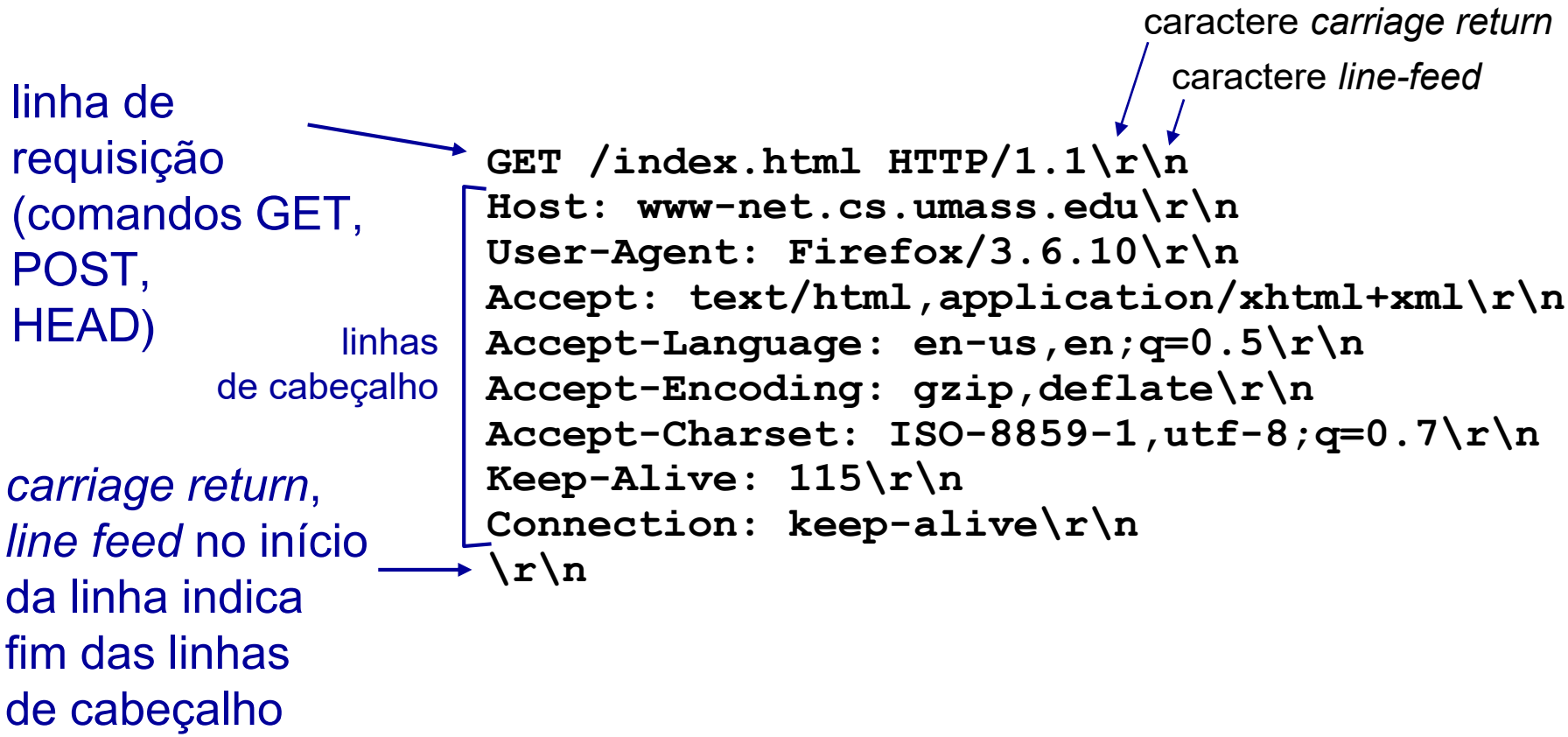
- usa 2 RTTs por objeto
- ocupa o SO a cada conexão TCP
- navegadores geralmente abrem conexões TCP paralelas para buscar objetos referenciados

HTTP persistente:

- servidor deixa a conexão aberta depois de enviar uma resposta
- mensagens HTTP subsequentes entre mesmo cliente/servidor enviadas sobre a conexão aberta
- cliente envia requisição assim que encontra um objeto referenciado
- apenas um RTT para cada objeto referenciado

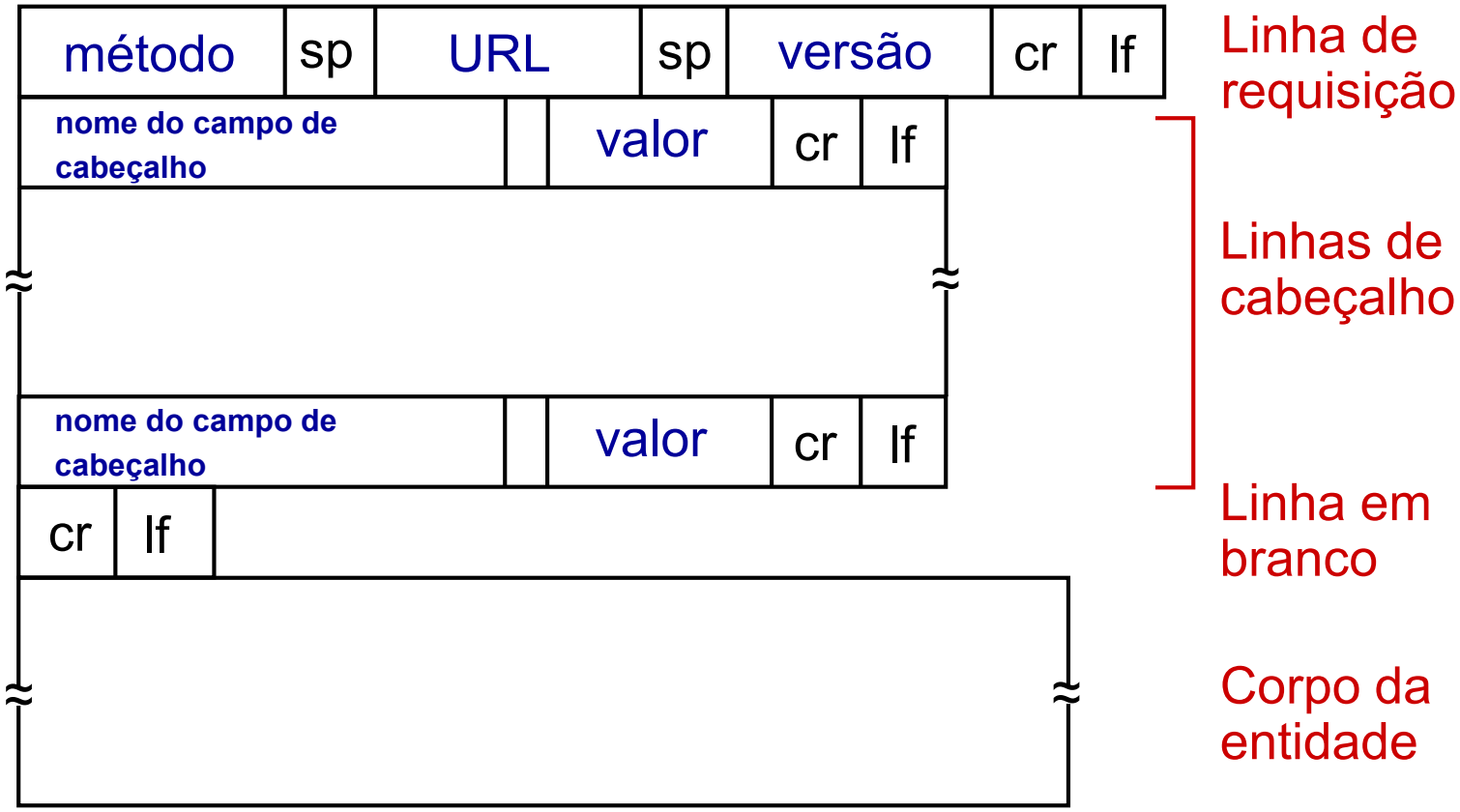
HTTP: Mensagem de Requisição

- dois tipos de mensagens HTTP: ***requisição*** e ***resposta***
- mensagem de requisição HTTP:
  - ASCII (formato legível por humanos)



HTTP: Mensagem de Requisição

Formato Geral



HTTP: Métodos de Entrada

método POST:

- página web inclui entrada por formulário
- entrada é carregada para o servidor no corpo da entidade

método URL:

- usa método GET
- entrada é carregada no campo URL da linha de requisição:

```
www.somesite.com/animalsearch?monkeys&banana
```

HTTP: Tipos de Métodos

HTTP/1.0:

- GET
- POST
- HEAD
  - similar ao GET, mas pede ao servidor para deixar o objeto solicitado de fora da resposta

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - faz upload de arquivo no corpo da entidade para o caminho especificado no campo URL
- DELETE
  - deleta arquivo especificado no campo URL

HTTP: Mensagem de Resposta

linha de estado (código de estado do protocolo, msg de estado)

linhas de cabeçalho

dados, ex.: arquivo HTML requisitado

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP: códigos de estado na resposta

- código de estado aparece na primeira linha da mensagem de resposta do servidor ao cliente
- alguns códigos de exemplo:

200 OK

- requisição bem-sucedida, o objeto requisitado está nesta mensagem

301 Moved Permanently

- objeto requisitado foi removido permanentemente, novo URL especificada nesta mensagem no campo Location

400 Bad Request

- mensagem de requisição não compreendida pelo servidor

404 Not Found

- documento requisitado não existe no servidor

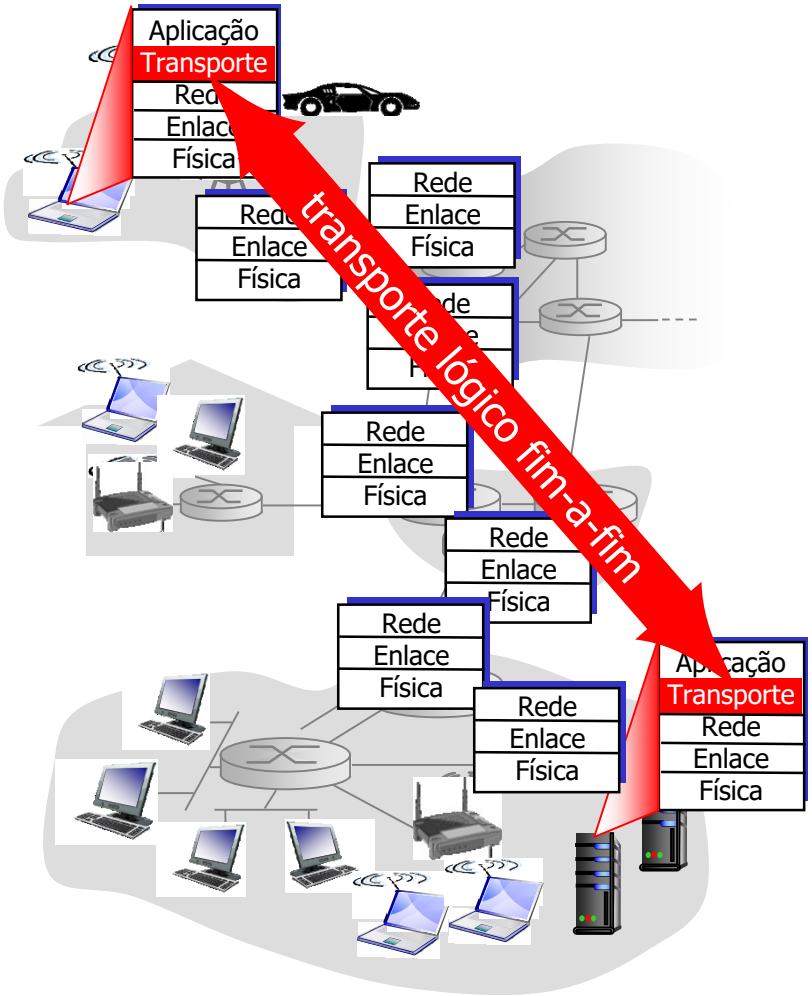
505 HTTP Version Not Supported

Sumário

- Introdução
- Camada de Aplicação
- **Camada de Transporte**

Camada de Transporte

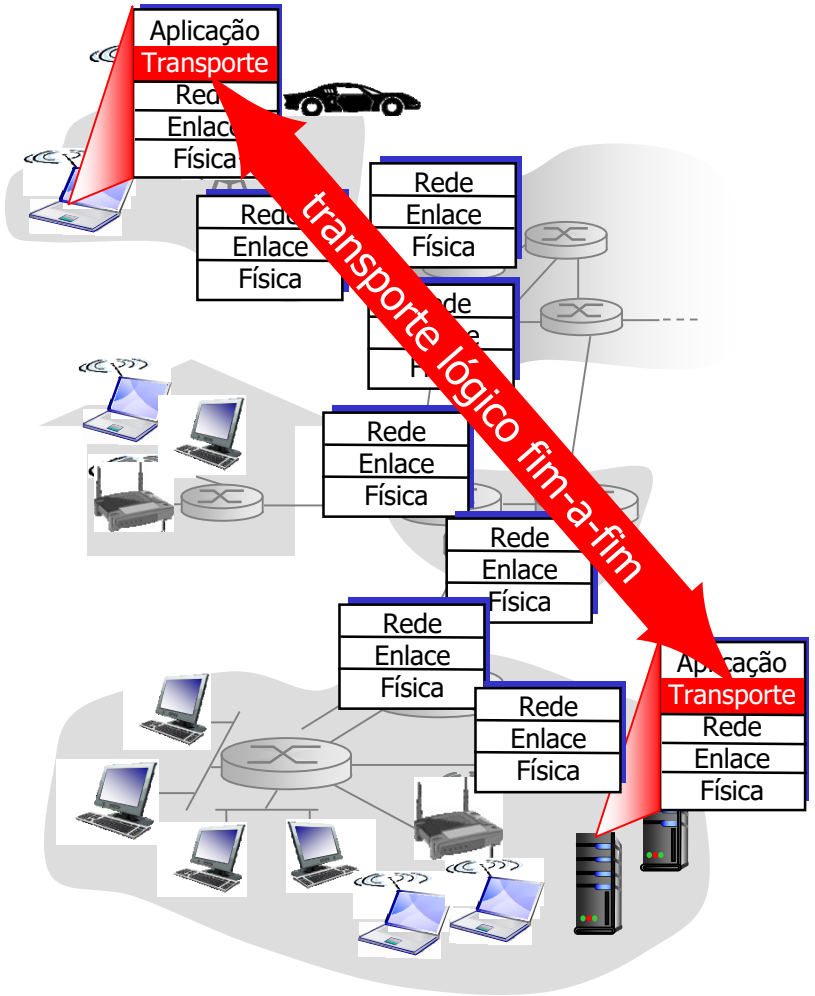
- **Camada de Transporte** provê *comunicação lógica* entre processos de aplicação rodando em diferentes hospedeiros
- protocolos de transporte rodam em sistemas finais
  - lado remetente: quebra mensagens em *segmentos*, repassa à camada de rede
  - lado receptor: remonta segmentos para formar mensagens, repassa à camada de aplicação
- mais de um protocolo
  - Internet: TCP e UDP



Camada de Transporte

Protocolos de Transporte da Internet

- confiável, entrega em ordem (TCP)
  - controle de congestionamento
  - controle de fluxo
  - estabelecimento de conexão
- não confiável, entrega fora de ordem (UDP)
  - camada de rede faz "melhor esforço" para entregar segmentos, mas sem garantia
- serviços não disponíveis
  - garantias de atraso
  - garantias de largura de banda



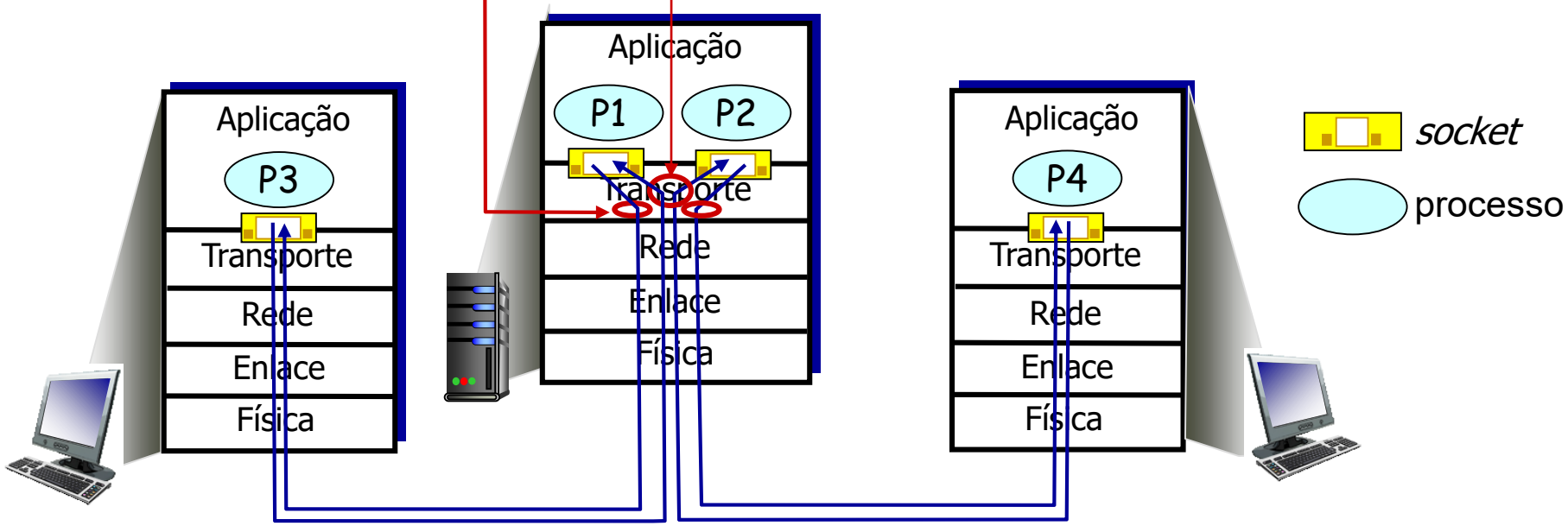
Multiplexação/Demultiplexação

multiplexação no remetente:

manipula dados de múltiplos *sockets*, adiciona cabeçalho de transporte (usado p/ demux)

demultiplexação no receptor:

usa info do cabeçalho para entregar segmento ao *socket* correto

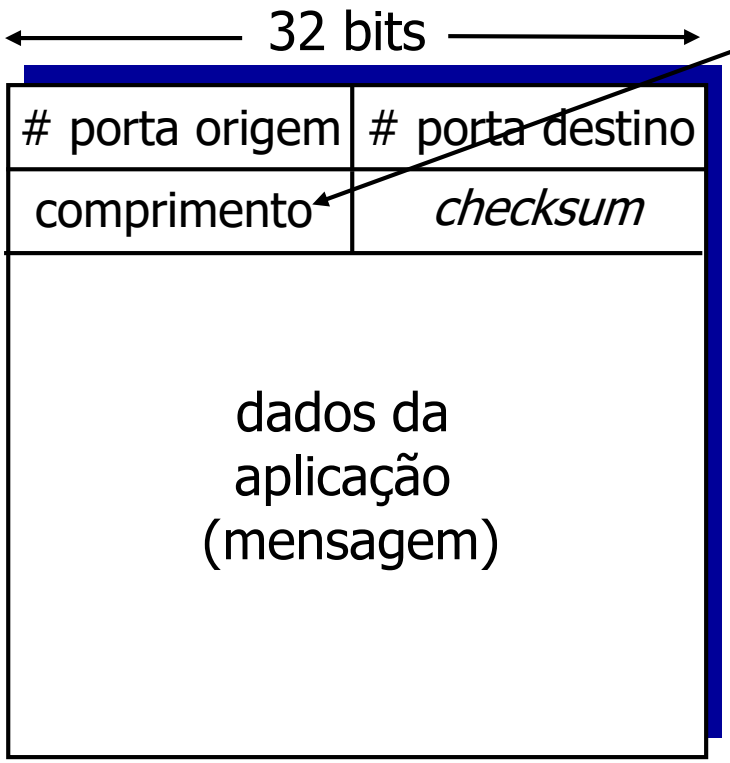


User Datagram Protocol (UDP)

- protocolo de transporte da Internet "sem cerimônias"
- serviço de "melhor esforço", segmentos UDP podem ser:
  - perdidos
  - entregues fora de ordem à aplicação
- *Sem conexão:*
  - sem apresentação entre remetente e receptor UDP
  - cada segmento UDP manipulado independentemente dos outros

- UDP usado em:
  - apps de *streaming* multimídia (tolera perda, sensível a taxa)
  - DNS
  - SNMP
- transferência confiável com UDP:
  - deve ser feita na camada de aplicação
  - recuperação de erro específica da aplicação!

UDP: Estrutura do Segmento



formato do segmento UDP

comprimento, em bytes do segmento UDP, incluindo cabeçalho

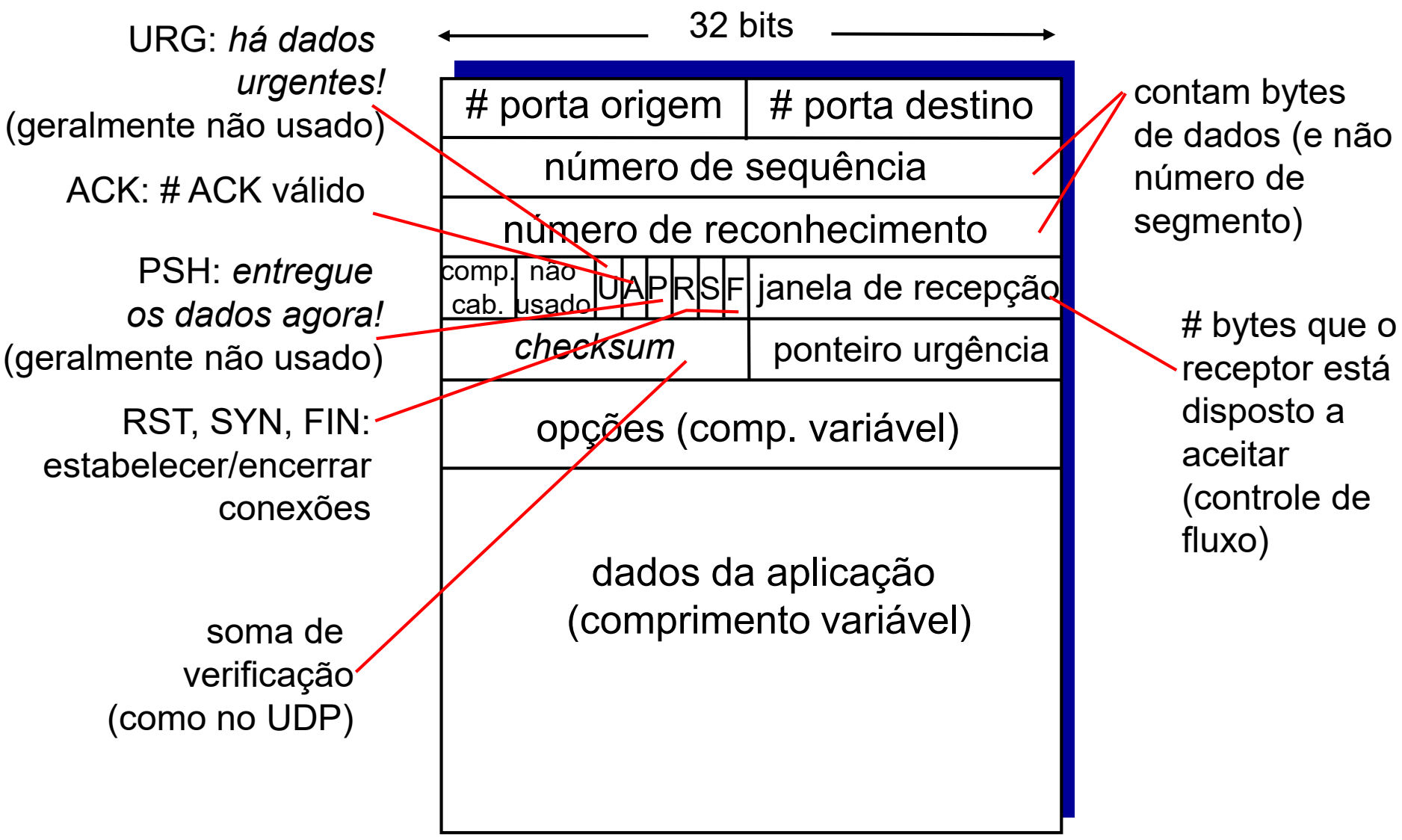
Por que existe UDP?

- sem estabelecimento de conexão (que pode adicionar atraso)
- simples: sem estado de conexão no remetente e receptor
- cabeçalho pequeno
- sem controle de congestionamento

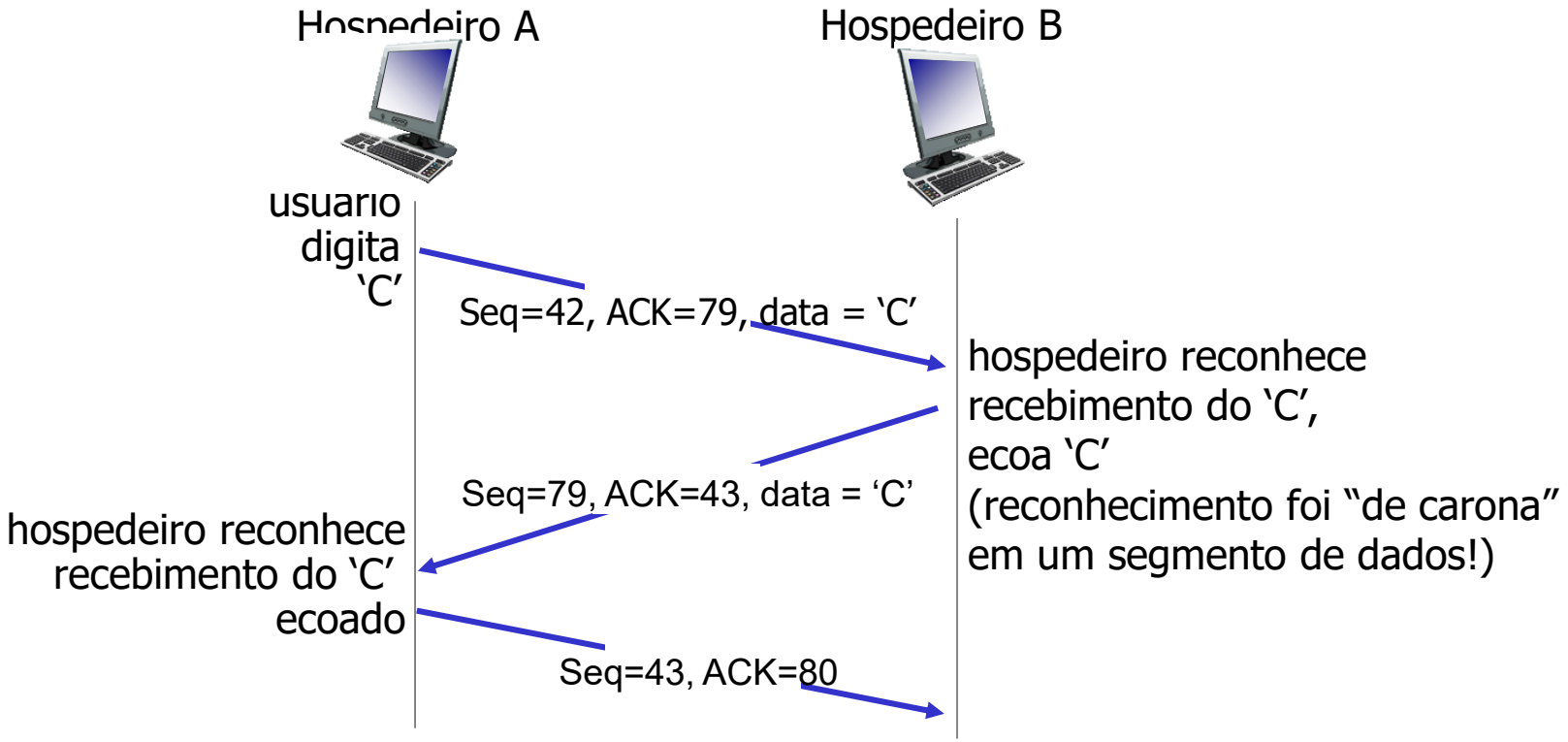
TCP: Visão Geral

- ponto a ponto:
  - um remetente, um receptor
- confiável, cadeia de bytes em ordem
  - dados recebidos na aplicação são idênticos aos enviados
- *pipelined*:
  - controles de congestionamento e de fluxo no TCP ajustam o tamanho da janela
- serviço *full-duplex*:
  - fluxo de dados flui bidirecionalmente na mesma conexão
- MSS: *maximum segment size*
- orientado a conexão:
  - apresentação (troca de msgs de controle) antes da troca real de dados
- controle de fluxo:
  - remetente não bombardeia o receptor

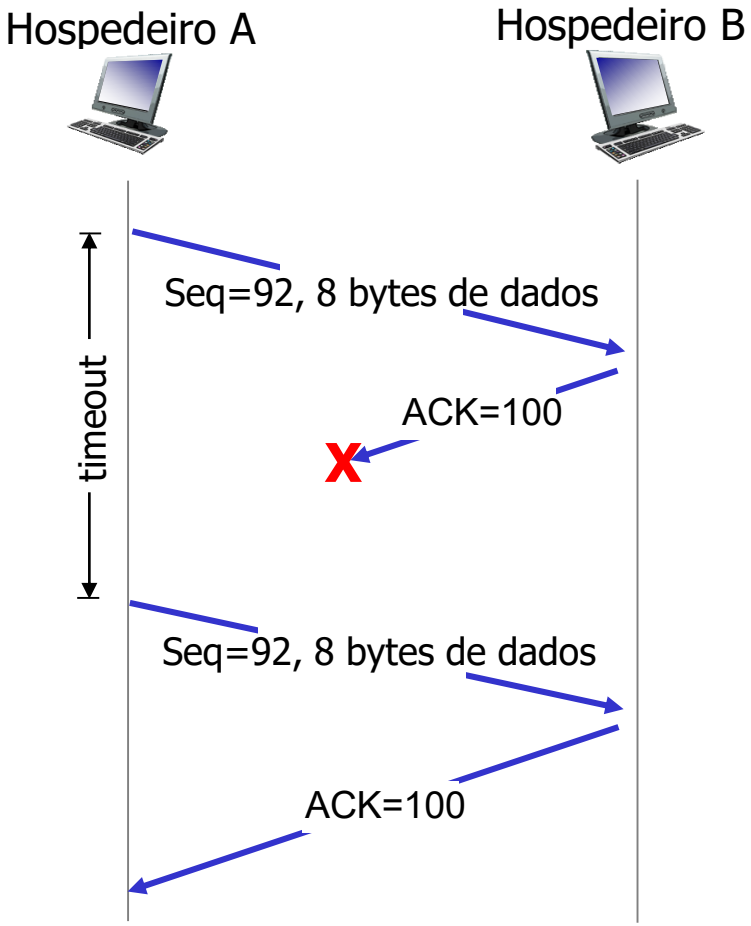
TCP: Estrutura do Segmento



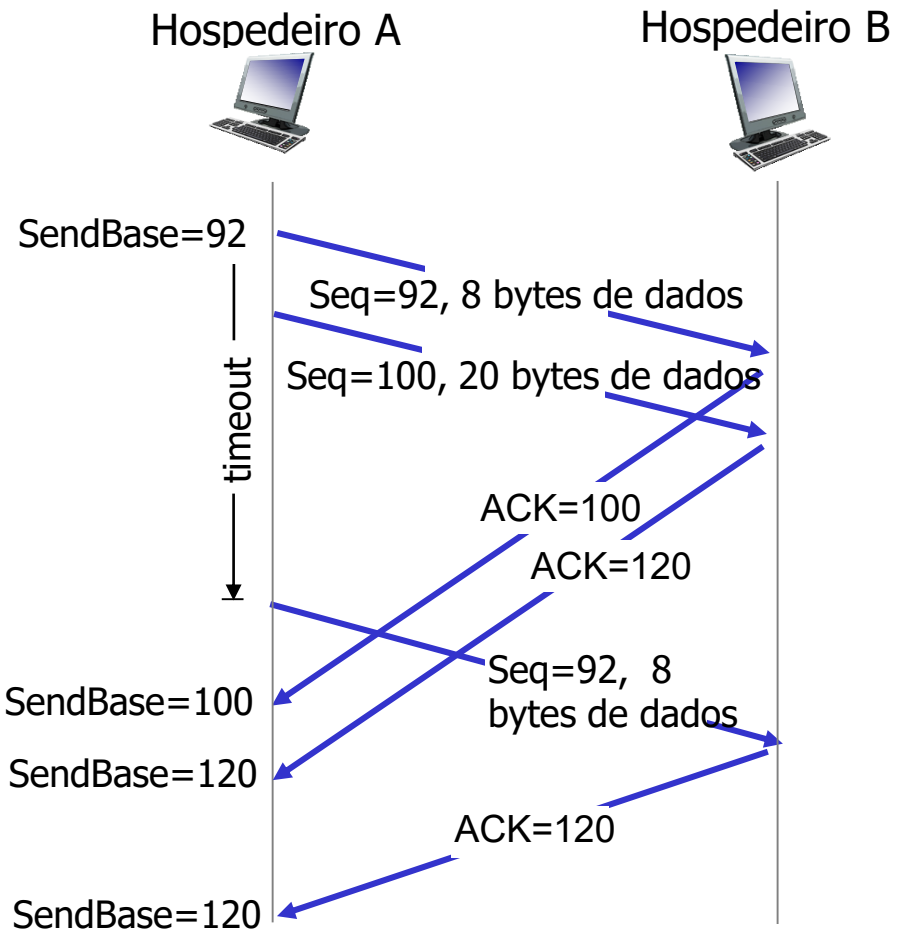
TCP: números de seq e de ACK



TCP: Cenários de Retransmissão



cenário de ACK perdido



timeout prematuro

TCP: Retransmissão Rápida

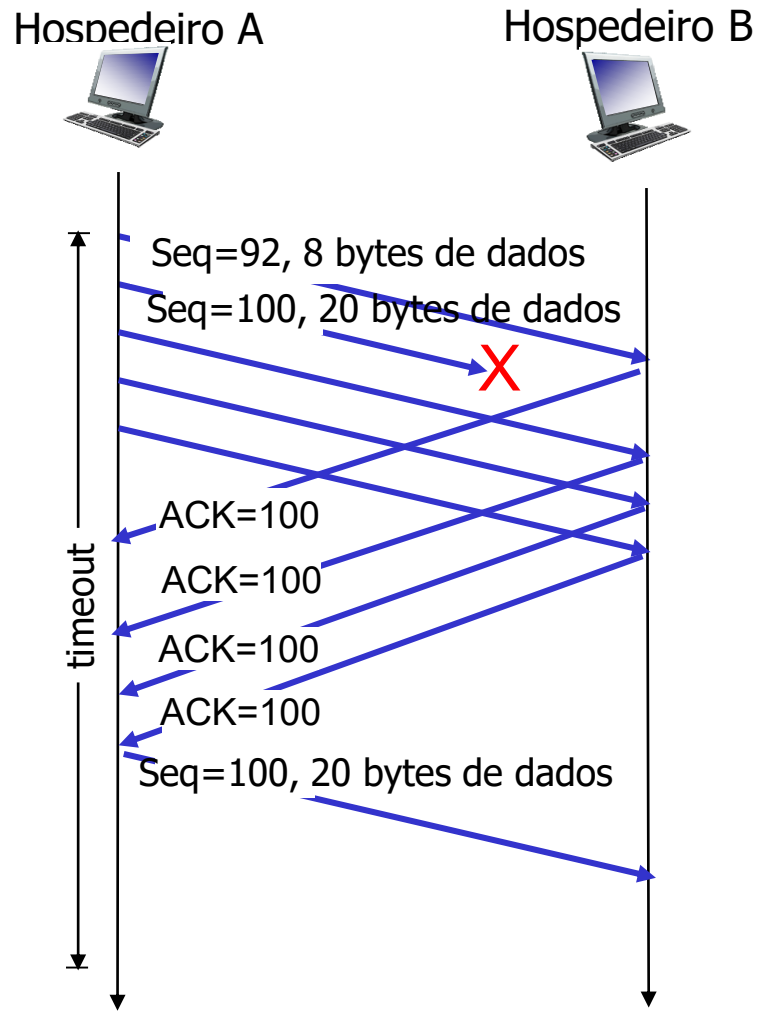
- intervalo de *timeout* relativamente longo frequentemente:
  - espera longa para reenviar pacote perdido
- detecta perda de segmentos via ACKs duplicados
  - muitos segmentos são geralmente enviados na sequência
  - se um segmento foi perdido, muitos ACKs duplicados vão chegar

Retransmissão Rápida

se remetente recebe 3 ACKs para os mesmos dados ("ACK triplamente repetido"), reenvia segmento não reconhecido com menor # de seq

- provavelmente perdido, não espera *timeout*

TCP: Retransmissão Rápida

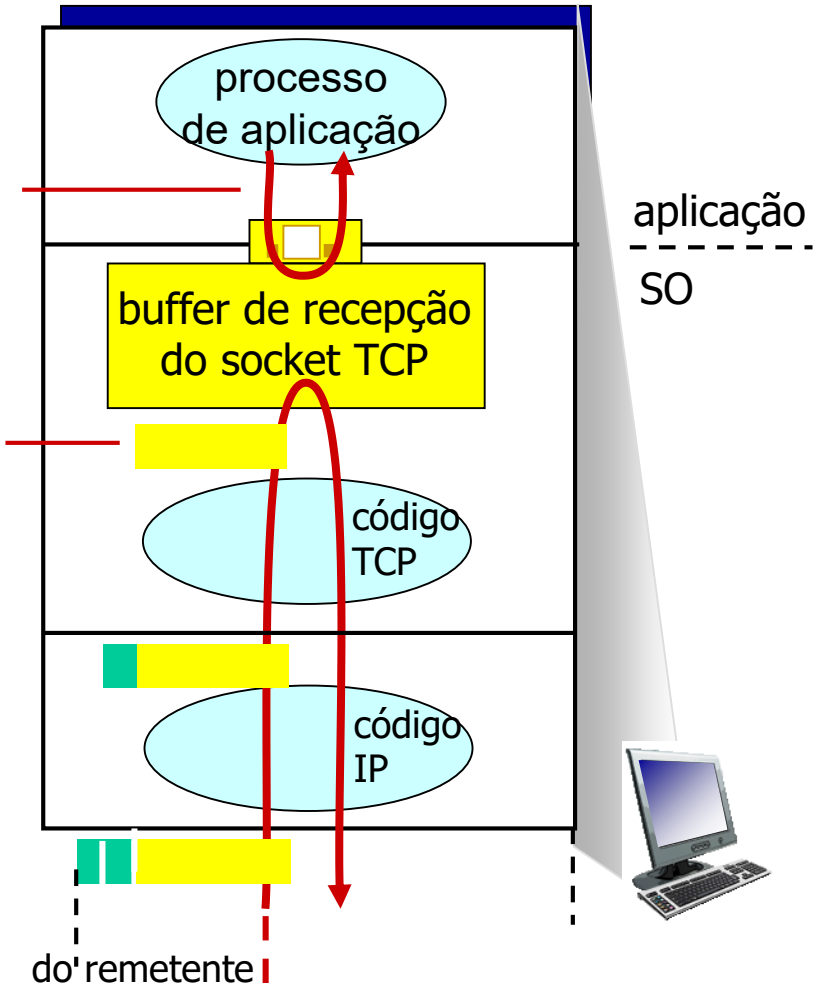


retransmissão rápida após remetente receber ACK triplamente repetido

TCP: Controle de Fluxo

aplicação pode remover dados do buffer de socket TCP....

... mais devagar que o receptor TCP está entregando (i.e., remetente está enviando)



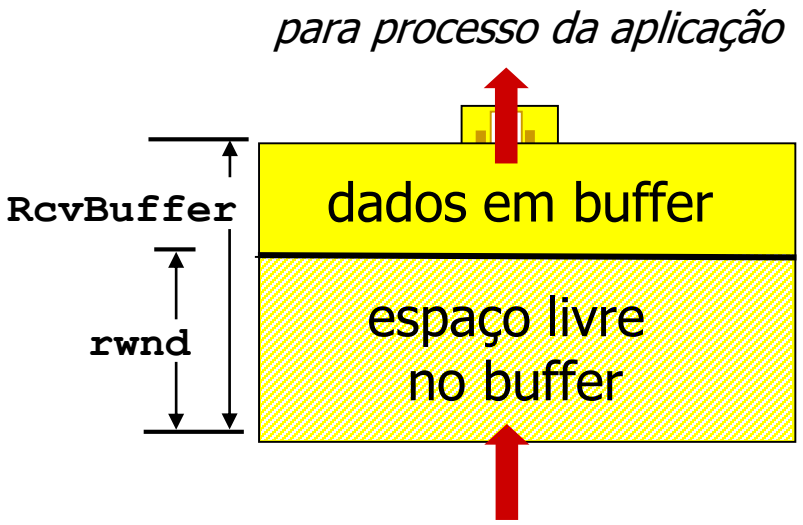
pilha de protocolos no receptor

controle de fluxo

receptor controla remetente para que este não transborde o buffer do receptor transmitindo demais e muito rápido

TCP: Controle de Fluxo

- receptor "avisa" espaço livre no buffer incluindo valor **`rwnd`** no cabeçalho dos segmentos TCP do receptor para o remetente
  - **`RcvBuffer`** tamanho configurado via opções do socket (tipicamente: 4096 bytes)
  - muitos SOs auto-ajustam **`RcvBuffer`**
- remetente limita número de dados não-ACKed ("em curso") de acordo com o valor de **`rwnd`**
- garante que buffer de recepção não transbordará

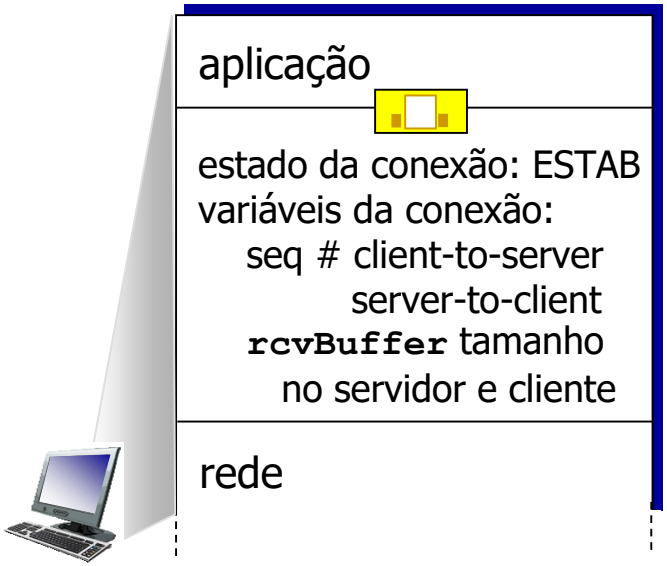


buffering no lado receptor

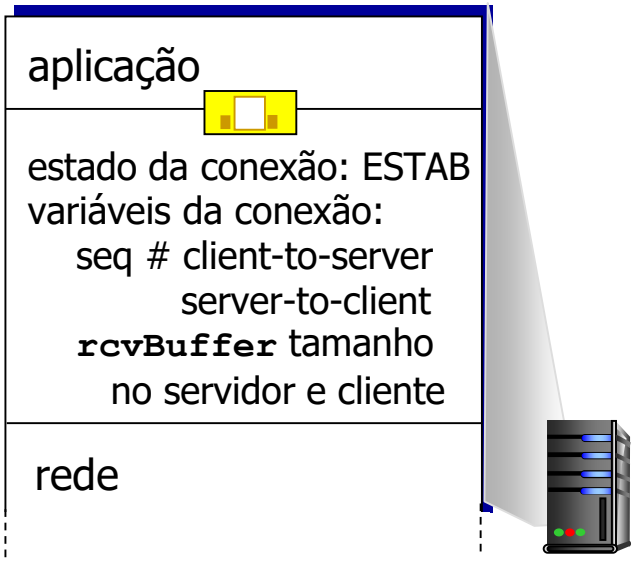
TCP: Gerenciamento de Conexão

antes de trocar dados, remetente/receptor "se apresentam":

- concordam em estabelecer conexão (ambos)
- concordam parâmetros da conexão

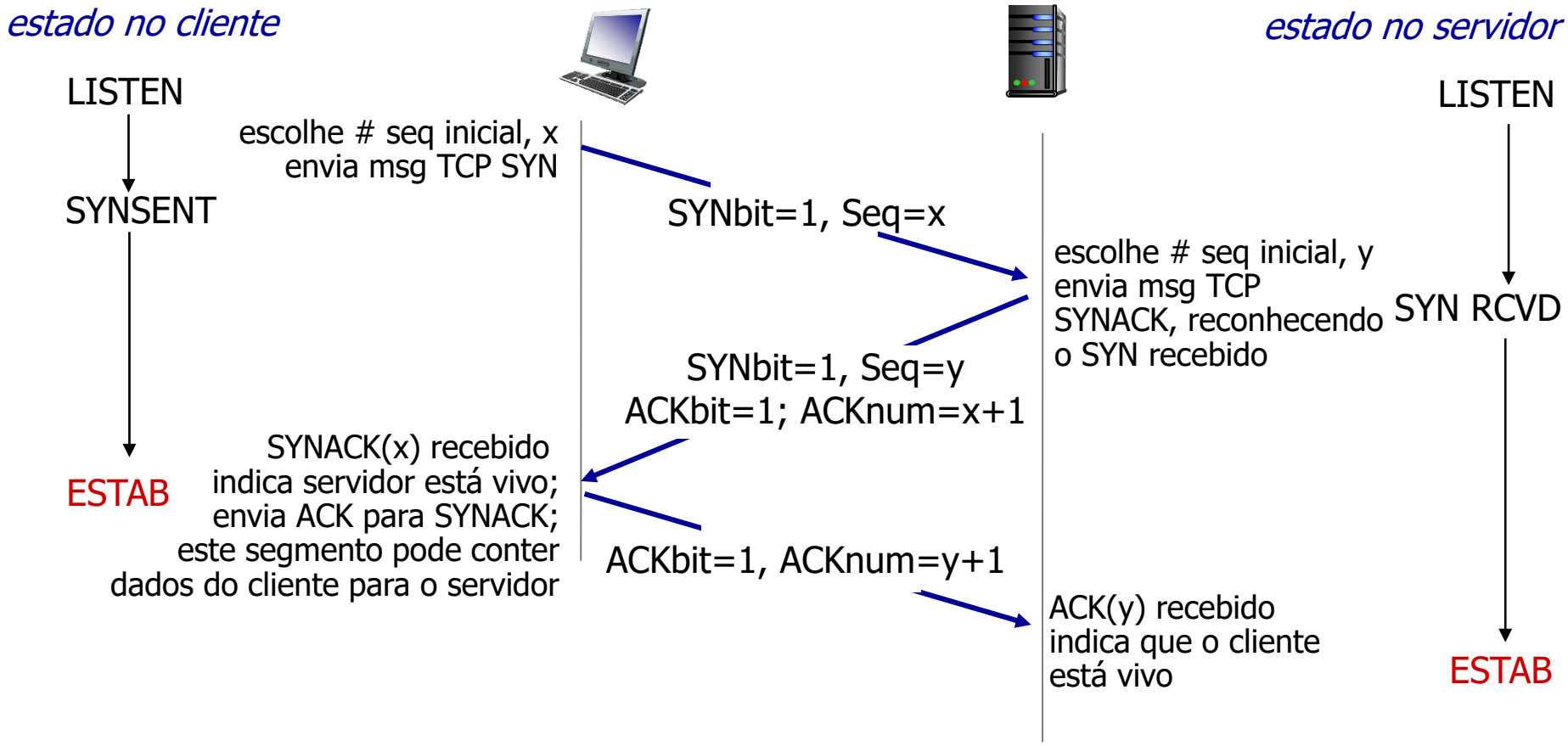



```
Socket clientSocket =
  newSocket("hostname","port
   number");
```




```
Socket connectionSocket =
  welcomeSocket.accept();
```

TCP: Apresentação de 3 Vias



TCP: Fechando uma Conexão

- ambos cliente e servidor fecham o seu lado da conexão
  - envia segmento TCP com bit FIN = 1
- respondem ao FIN recebido com ACK
  - ao receber FIN, ACK pode ser combinado com FIN próprio
- troca de FINs simultâneos pode ser feita

Abordagens para Controle de Congestionamento

dois mecanismos para controle de congestionamento:

controle de congest. fim-a-fim:

- sem *feedback* explícito da rede
- congest. inferido da perda e do atraso observados no sistema final
- abordagem usada pelo TCP

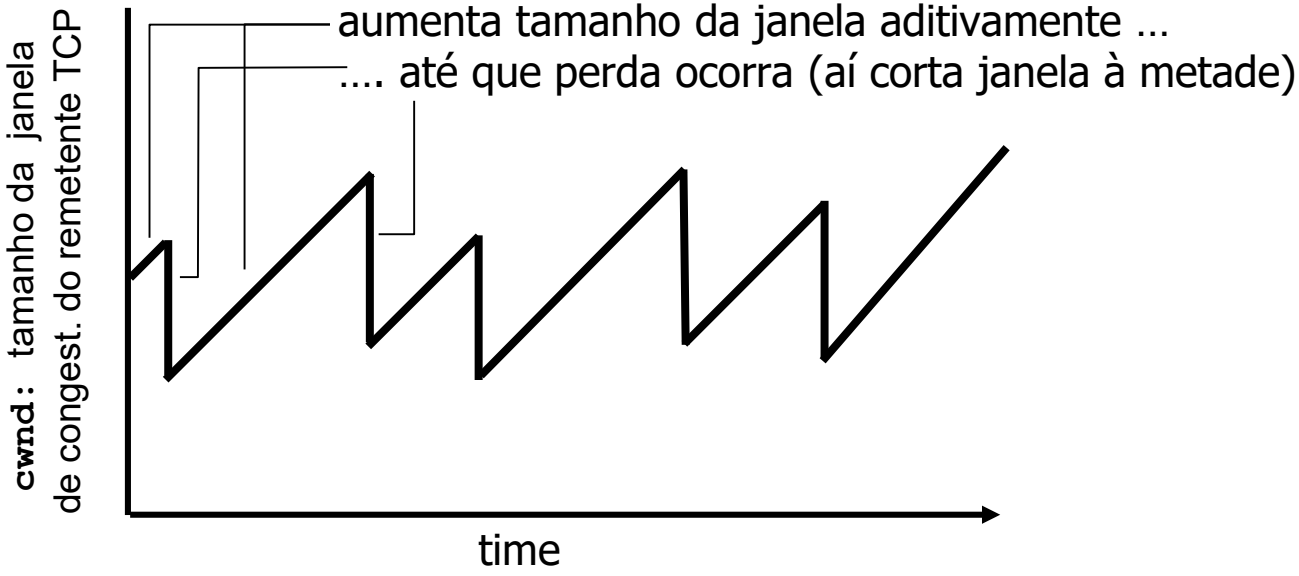
controle de congest. assistido pela rede:

- roteadores provêm *feedback* aos sist. finais
  - um bit indicando congestionamento (SNA, DECbit, TCP/IP ECN, ATM)
  - taxa explícita para o remetente enviar

TCP: Controle de Congestionamento

Aumento aditivo, Redução multiplicativa

- *abordagem:* remetente aumenta taxa de transmissão (janela) até que perda ocorra
  - *aumento aditivo:* aumento de **`cwnd`** em 1 MSS a cada RTT até que perda seja detectada
  - *redução multiplicativa:* reduz **`cwnd`** à metade após perda



UNIVERSIDADE FEDERAL DE PELOTAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

TEC2/TEC4: REDES MULTIMÍDIA (RMM)

Unidade 7
Revisão de Redes de Computadores
Camadas de Aplicação e Transporte

Prof. Guilherme Corrêa
gcorrea@inf.ufpel.edu.br