Capítulo 2: Camada de aplicação

REDES DE COMPUTADORES E A INTERNET 5ª edição
Uma Abordagem Top-Down

Objetivos do capítulo:

- aspectos conceituais, de implementação de protocolos de aplicação de rede
  - modelos de serviço da camada de transporte
  - paradigma cliente-servidor
  - paradigma *peer-to-peer*
- aprenda sobre protocolos examinando protocolos populares em nível de aplicação
  - HTTP
  - FTP
  - SMTP/POP3/IMAP
  - DNS
- programando aplicações de rede
  - API socket

Algumas aplicações de rede

- e-mail
- web
- mensagem instantânea
- login remoto
- compartilhamento de arquivos P2P
- jogos em rede multiusuários
- clipes de vídeo armazenados em fluxo contínuo
- redes sociais
- voice over IP
- vídeoconferência em tempo real
- computação em grade

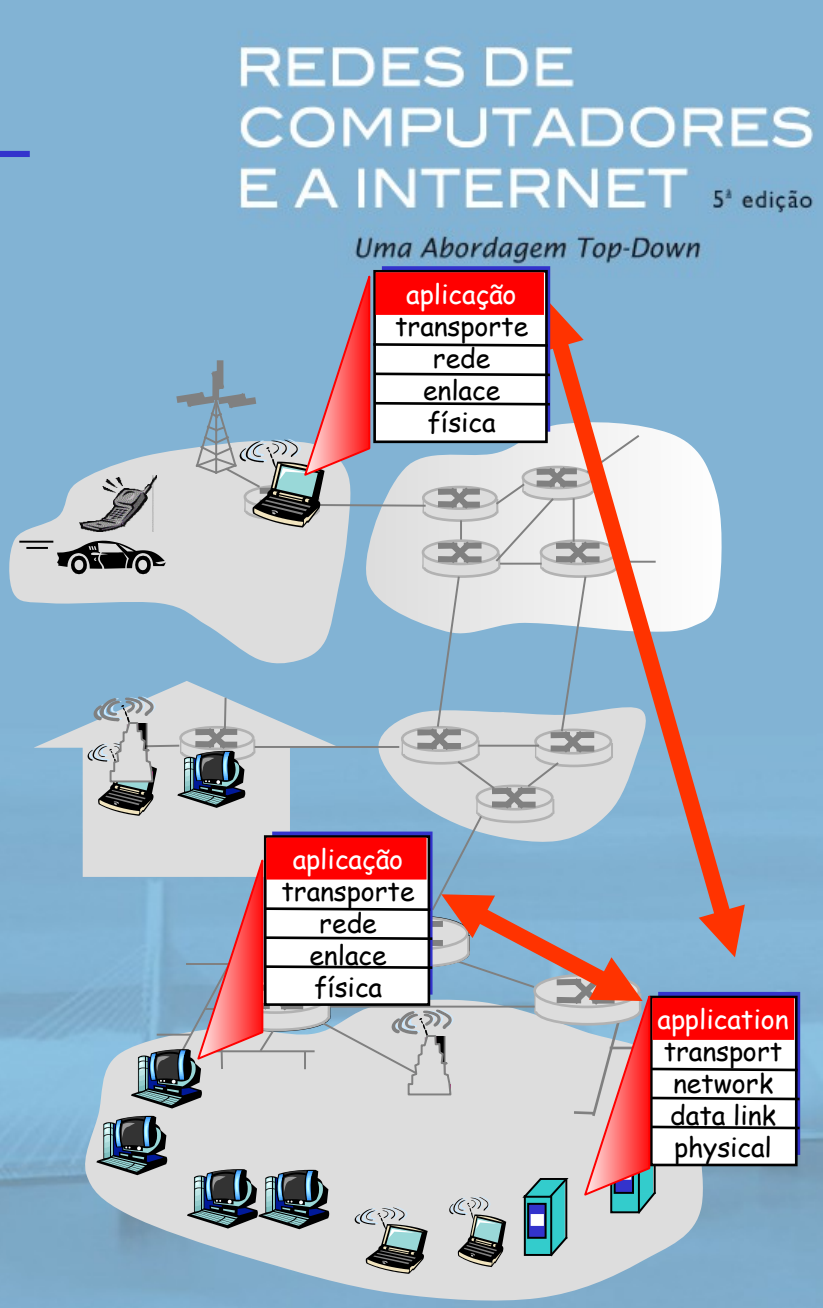
Criando uma aplicação de rede

Escreva programas que

- executem em (diferentes) *sistemas finais*
- se comuniquem pela rede
- p. e., software de servidor Web se comunica com software de navegador Web

Não é preciso escrever software para dispositivos do núcleo da rede

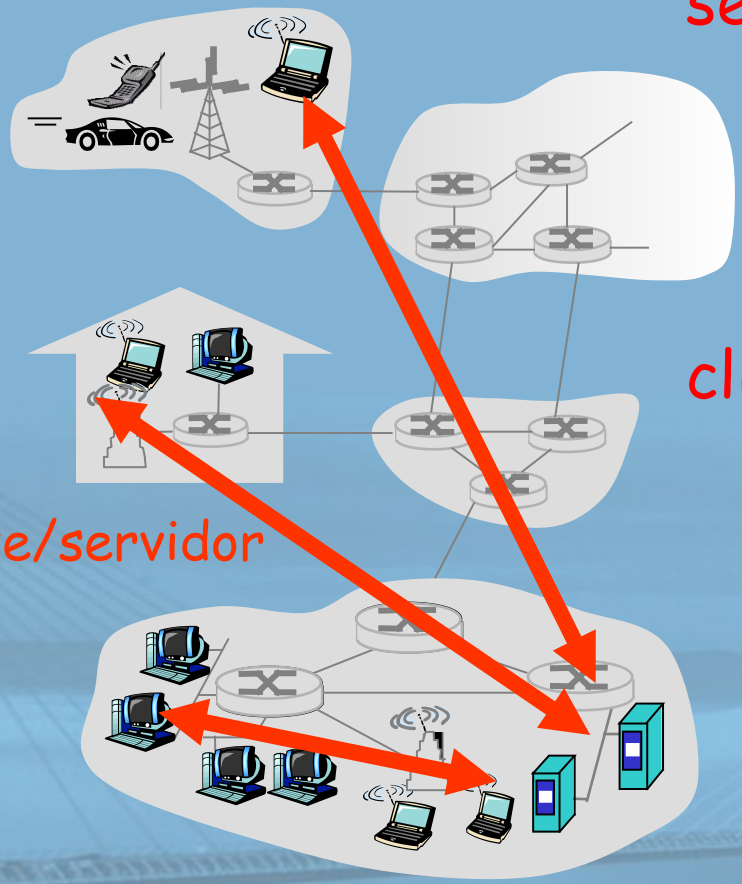
- dispositivos do núcleo da rede não executam aplicações do usuário
- as aplicações nos sistemas finais permitem rápido desenvolvimento e propagação



Arquiteturas de aplicação

- Cliente-servidor
  - Incluindo centros de dados/cloud computing
- Peer-to-peer (P2P)
- Híbrida de cliente-servidor e P2P

Arquitetura cliente-servidor



servidor:

- hospedeiro sempre ligado
- endereço IP permanente

clientes:

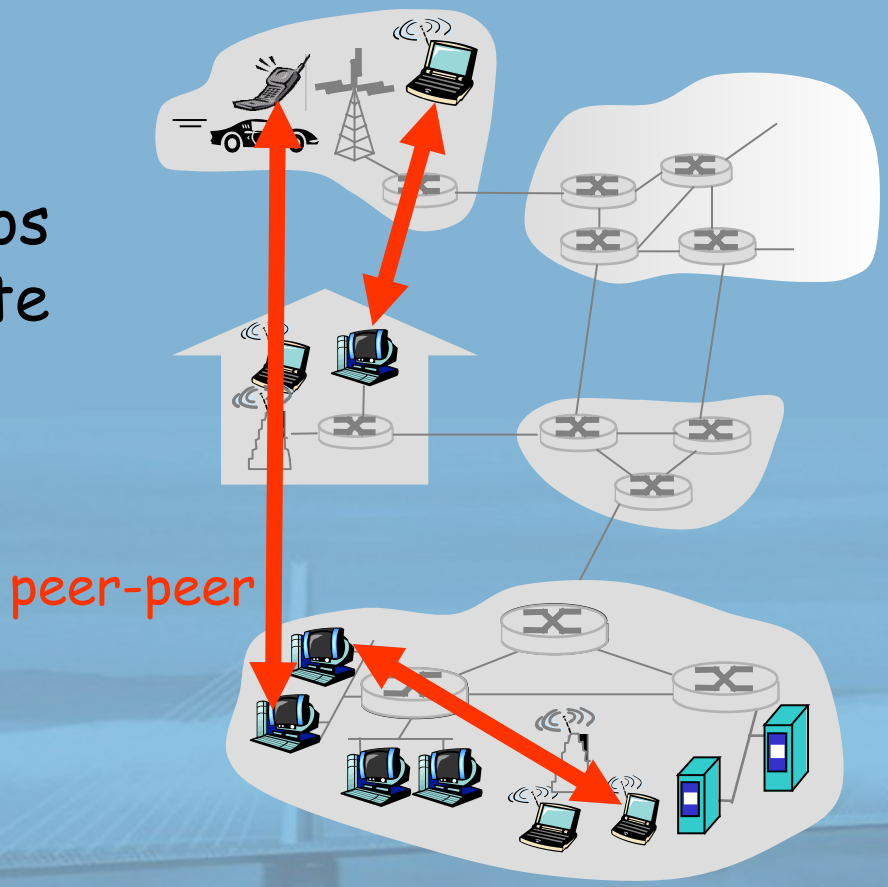
- comunicam-se com o servidor
- podem estar conectados intermitentemente
- podem ter endereços IP dinâmicos
- não se comunicam diretamente entre si

Centros de dados da Google

- custo estimado do centro de dados: $600M
- Google gastou $2,4B em 2007 em novos centros de dados
- cada centro de dados usa de 50 a 100 megawatts de potência

Arquitetura P2P pura

- *nenhum* servidor sempre ligado
- sistemas finais arbitrários se comunicam diretamente
- pares são conectados intermitentemente e mudam endereços IP



altamente escalável, mas difícil de administrar

Híbrido de cliente-servidor e P2P

Skype

- aplicação P2P voice-over-IP P2P
- servidor centralizado: achando endereço da parte remota:
- conexão cliente-cliente: direta (não através de servidor)

Mensagem instantânea

- bate-papo entre dois usuários é P2P
- serviço centralizado: detecção/localização da presença do cliente
  - usuário registra seu endereço IP com servidor central quando entra on-line
  - usuário contacta servidor central para descobrir endereços IP dos parceiros

Processos se_comunicando

processo: programa rodando dentro de um hospedeiro

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

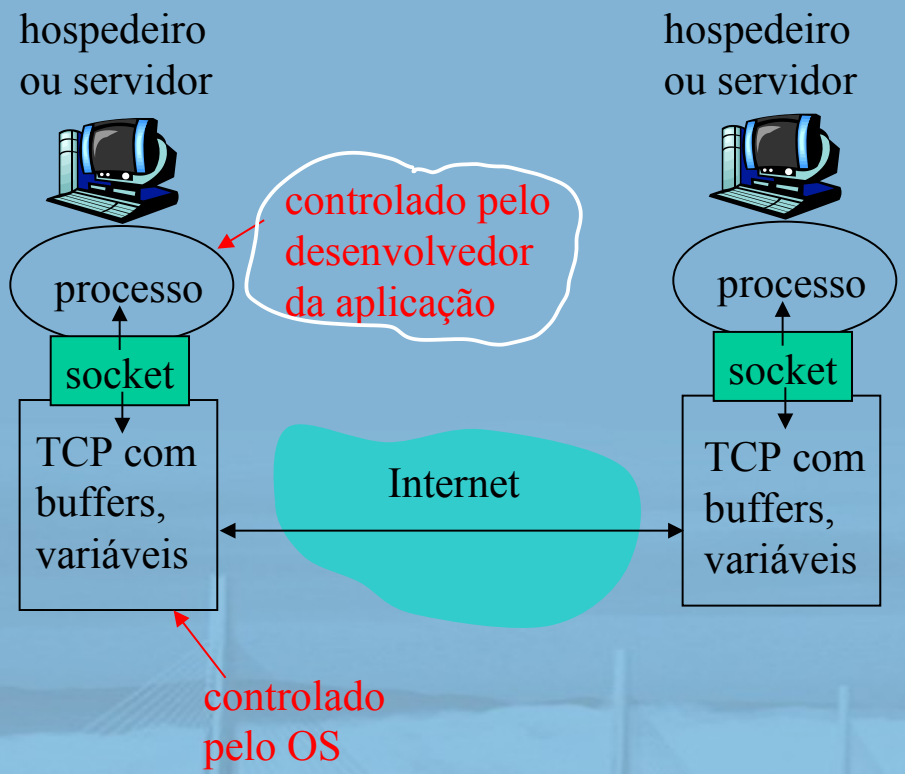
processo cliente: processo que inicia a comunicação

processo servidor: processo que espera para ser contactado

- Nota: aplicações com arquiteturas P2P têm processos clientes & processos servidores

Sockets

- processo envia/recebe mensagens de/para seu socket
- socket semelhante à porta
  - processo enviando empurra mensagem pela porta
  - processo enviando conta com infraestrutura de transporte no outro lado da porta, que leva a mensagem ao socket no processo receptor



Endereçando processos

- para receber mensagens, processo deve ter *identificador*
- dispositivo hospedeiro tem endereço IP exclusivo de 32 bits
- exercício: use `ipconfig` do comando prompt para obter seu endereço IP (Windows)
- *P:* Basta o endereço IP do hospedeiro em que o processo é executado para identificar o processo?
  - *R:* Não, *muitos* processos podem estar rodando no mesmo hospedeiro
- *Identificador* inclui endereço IP e números de porta associados ao processo no hospedeiro.
- Exemplos de número de porta:
  - servidor HTTP: 80
  - servidor de correio: 25

Definições de protocolo da camada de aplicação

- tipos de mensagens trocadas,
  - p. e., requisição, resposta
- sintaxe da mensagem:
  - que campos nas mensagens & como os campos são delineados
- semântica da mensagem
  - significado da informação nos campos
- regras de quando e como processos enviam & respondem a mensagens

protocolos de domínio público:

- definidos em RFCs
- provê interoperabilidade
- p. e., HTTP, SMTP, BitTorrent

protocolos proprietários:

- p. e., Skype, ppstream

Que serviço de transporte uma aplicação precisa?

perda de dados

- algumas apls. (p. e., áudio) podem tolerar alguma perda
- outras apls. (p. e., transferência de arquivos, telnet) exigem transferência de dados 100% confiável

temporização

- algumas apls. (p. e., telefonia na Internet jogos interativos) exigem pouco atraso para serem "eficazes"

vazão

- algumas apls. (p. e., multimídia) exigem um mínimo de vazão para serem "eficazes"
- outras apls. ("apls. elásticas") utilizam qualquer vazão que receberem

segurança

- criptografia, integridade de dados,...

Requisitos de serviço de transporte das aplicações comuns

Aplicação	Perda de dados	Vazão	Sensível ao tempo
transf. arquivos	sem perda	elástica	não
e-mail	sem perda	elástica	não
documentos Web	sem perda	elástica	não
áudio/vídeo tempo real	tolerante a perda	áudio: 5 kbps-1 Mbps vídeo:10 kbps-5 Mbps	sim, centenas de ms
áudio/vídeo armazenado	tolerante a perda	o mesmo que antes	sim, alguns seg
jogos interativos	tolerante a perda	poucos kbps ou mais	sim, centenas de ms
Mensagem instantânea	sem perda	elástica	sim e não

Serviços de protocolos de transporte da Internet

serviço TCP:

- *orientado a conexão:* preparação exigida entre processos cliente e servidor
- *transporte confiável* entre processo emissor e receptor
- *controle de fluxo:* emissor não sobrecarrega receptor
- *controle de congestionamento:* regula emissor quando a rede está sobrecarregada
- *não oferece:* temporização, garantias mínimas de vazão, segurança

serviço UDP:

- transferência de dados não confiável entre processo emissor e receptor
- não oferece: preparação da conexão, confiabilidade, controle de fluxo, controle de congest., temporização, garantia de vazão ou segurança

Aplicações da Internet: aplicação, protocolos de transporte

Aplicação	Protocolo da camada de aplicação	Protocolo de transporte básico
e-mail	SMTP [RFC 2821]	TCP
acesso remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transf. arquivos	FTP [RFC 959]	TCP
multimídia com fluxo contínuo	HTTP (p. e., Youtube), RTP [RFC 1889]	TCP ou UDP
telefonia da Internet	SIP, RTP, proprietário (p. e., Skype)	normalmente UDP

Web e HTTP

primeiro, algum jargão

- página Web consiste em objetos
- objeto pode ser arquivo HTML, imagem JPEG, applet Java, arquivo de áudio,...
- página Web consiste em arquivo HTML básico que inclui vários objetos referenciados
- cada objeto é endereçável por um URL
- exemplo de URL:

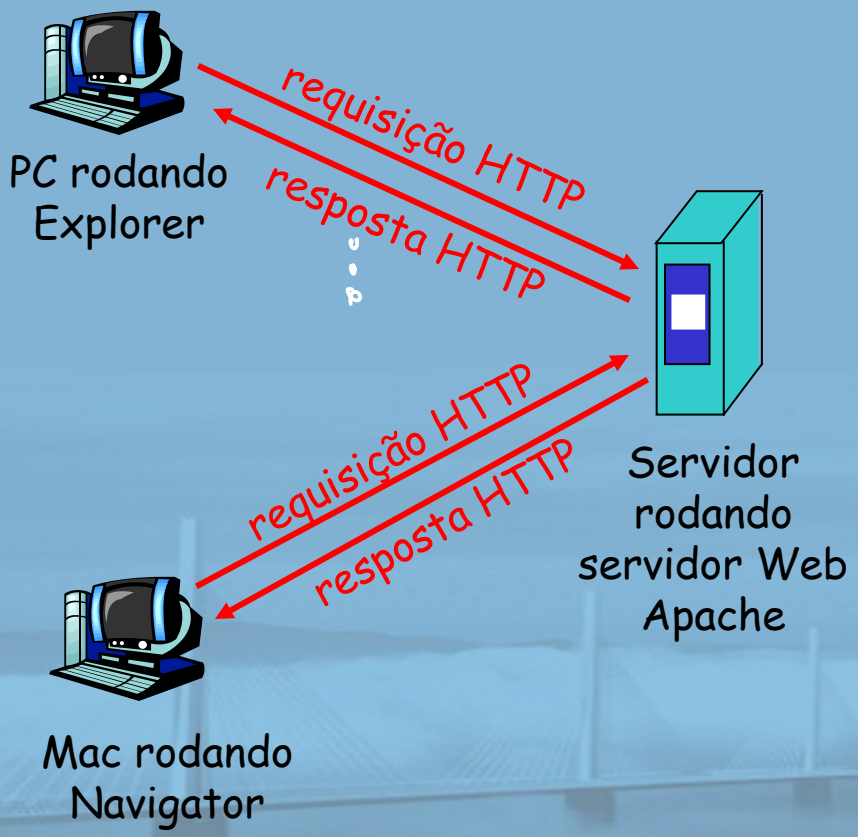
`www.someschool.edu/someDept/pic.gif`

nome do hospedeiro | nome do caminho

Visão geral do HTTP

HTTP: HyperText Transfer Protocol

- protocolo da camada de aplicação da Web
- modelo cliente/servidor
  - *cliente:* navegador que requisita, recebe, "exibe" objetos Web
  - *servidor:* servidor Web envia objetos em resposta a requisições



usa TCP:

- cliente inicia conexão TCP (cria socket) com servidor, porta 80
- servidor aceita conexão TCP do cliente
- mensagens HTTP (do protocolo da camada de aplicação) trocadas entre navegador (cliente HTTP) e servidor Web (servidor HTTP)
- conexão TCP fechada

HTTP é "sem estado"

- servidor não guarda informações sobre requisições passadas do cliente

aparte

Protocolos que mantêm "estado" são complexos!

- história passada (estado) deve ser mantida
- se servidor/cliente falhar, suas visões do "estado" podem ser incoerentes, devem ser reconciliadas

Conexões HTTP

HTTP não persistente

- no máximo um objeto é enviado por uma conexão TCP.

HTTP persistente

- múltiplos objetos podem ser enviados por uma única conexão TCP entre cliente e servidor.

HTTP não persistente

Suponha que o usuário digite o URL `www.someSchool.edu/someDepartment/home.index`
(contém texto, referências a 10 imagens JPEG)

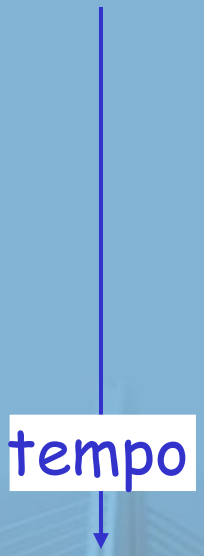
1a. Cliente HTTP inicia conexão TCP com servidor HTTP (processo) em www.someSchool.edu na porta 80.

1b. Servidor HTTP no hospedeiro www.someSchool.edu esperando conexão TCP na porta 80. "aceita" conexão, notificando cliente

2. Cliente HTTP envia *mensagem de requisição* HTTP (contendo URL) pelo socket de conexão TCP. Mensagem indica que cliente deseja o objeto someDepartment/home.index.

3. Servidor HTTP recebe mensagem de requisição, forma *mensagem de resposta* contendo objeto requisitado e envia mensagem para seu socket

tempo



4. Servidor HTTP fecha conexão TCP.

5. Cliente HTTP recebe mensagem de resposta contendo arquivo html, exibe html. Analisando arquivo html, acha 10 objetos JPEG referenciados.

6. Etapas 1-5 repetidas para cada um dos 10 objetos JPEG.

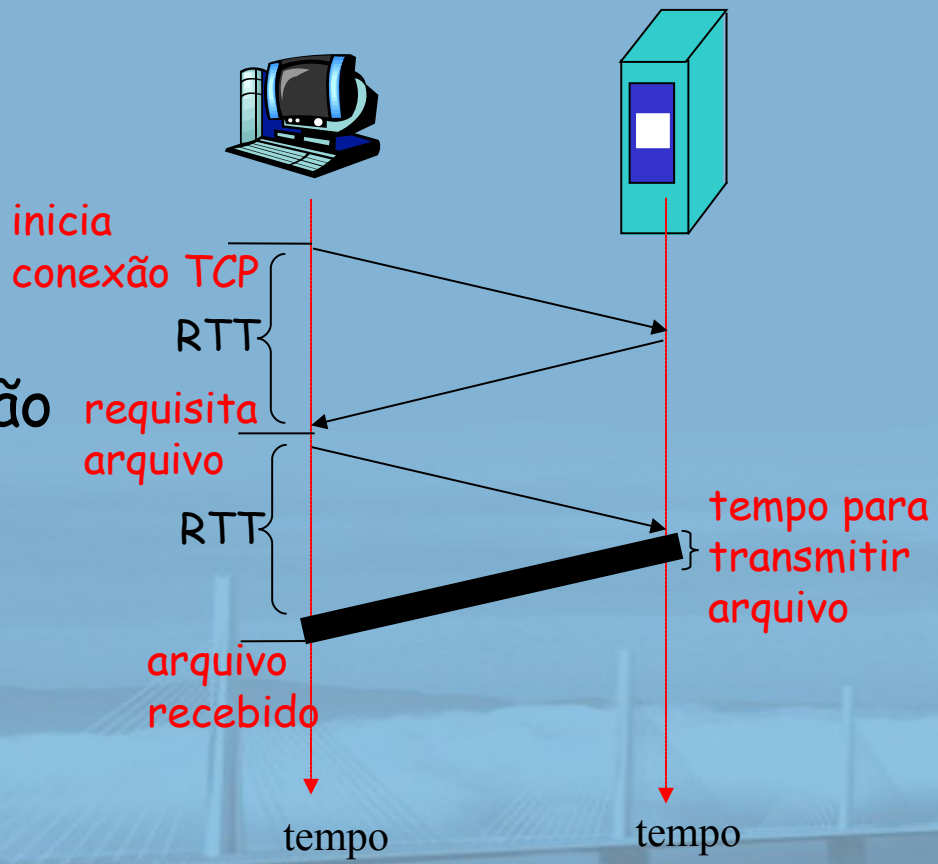
HTTP não persistente: tempo de resposta

definição de RTT: tempo para um pequeno pacote trafegar do cliente ao servidor e retornar.

tempo de resposta:

- um RTT para iniciar a conexão TCP
- um RTT para a requisição HTTP e primeiros bytes da resposta HTTP retornarem
- tempo de transmissão de arquivo

total = 2RTT + tempo de transmissão



HTTP persistente

problemas do HTTP não persistente:

- requer 2 RTTs por objeto
- overhead do SO para *cada* conexão TCP
- navegadores geralmente abrem conexões TCP paralelas para buscar objetos referenciados

HTTP persistente:

- servidor deixa a conexão aberta depois de enviar a resposta
- mensagens HTTP seguintes entre cliente/servidor enviadas pela conexão aberta
- cliente envia requisições assim que encontra um objeto referenciado
- no mínimo um RTT para todos os objetos referenciados

Mensagem de requisição HTTP

- dois tipos de mensagens HTTP: *requisição, resposta*
- mensagem de requisição HTTP:
  - ASCII (formato de texto legível)

linha de requisição (comandos GET, POST, HEAD)

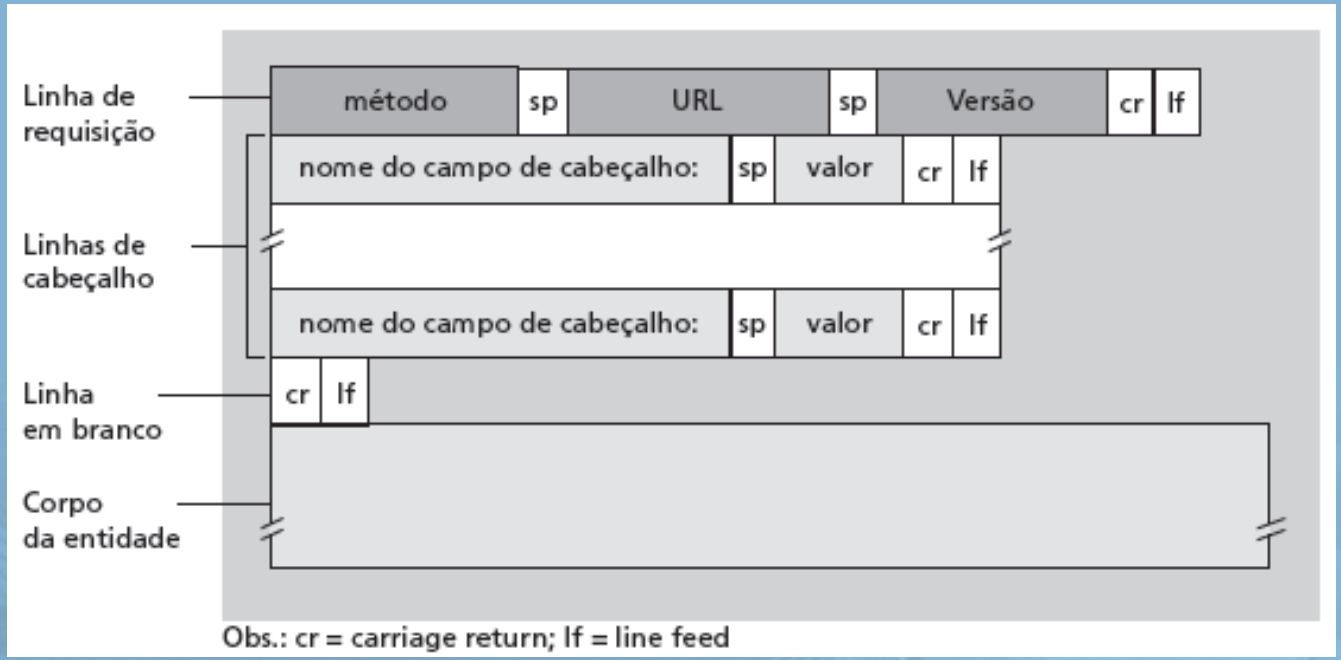
linhas de cabeçalho

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

carriage return, line feed indica final da mensagem

(carriage return, line feed extras)

Mensagem de requisição HTTP: formato geral



Obs.: cr = carriage return; lf = line feed

Upload da entrada do formulário

método POST:

- página Web geralmente inclui entrada do formulário
- entrada é enviada ao servidor no corpo da entidade

método do URL:

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

```
www.umsite.com/buscaanimal?macacos&banana
```

Tipos de método

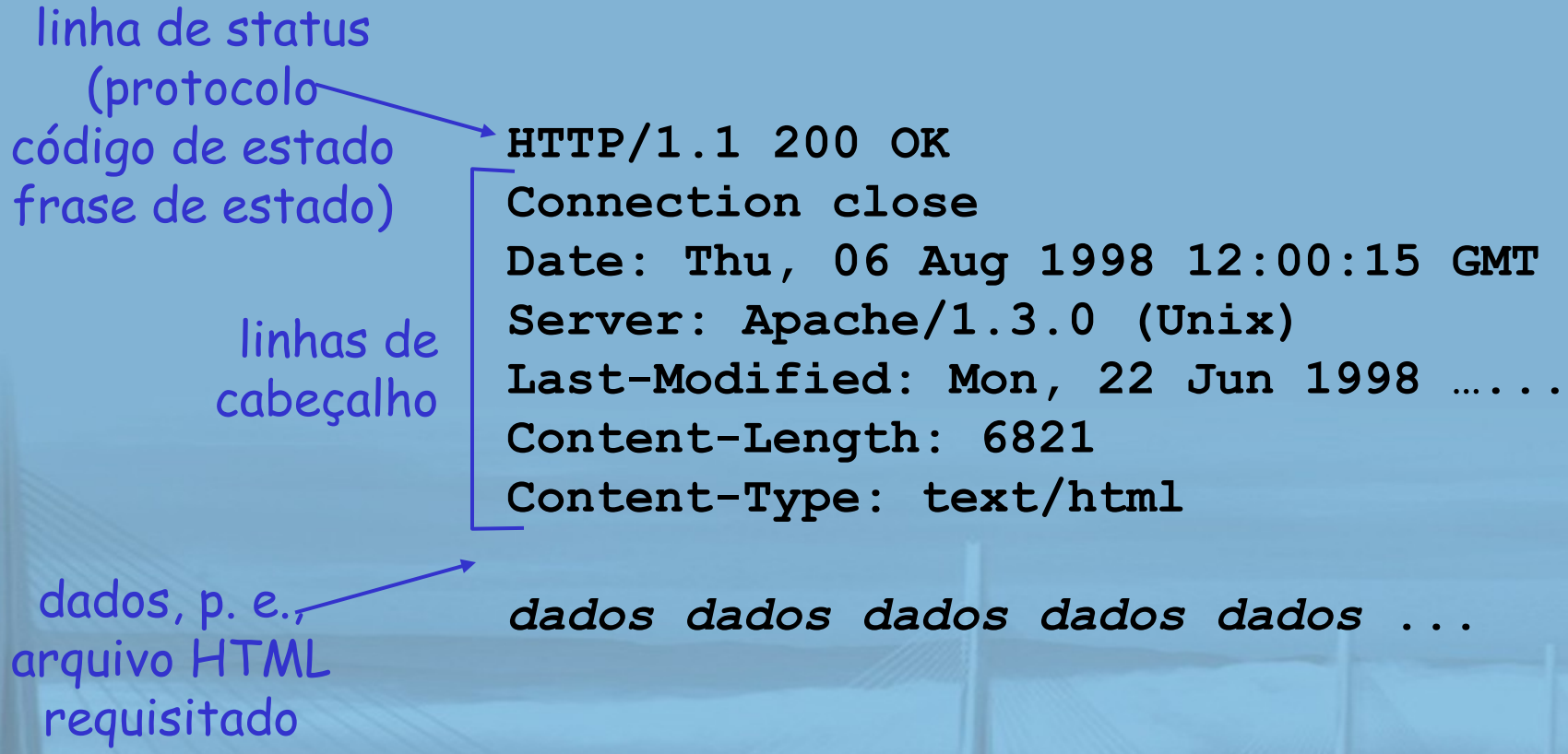
HTTP/1.0

- GET
- POST
- HEAD
  - pede ao servidor para deixar objeto requisitado fora da resposta

HTTP/1.1

- GET, POST, HEAD
- PUT
  - envia arquivo no corpo da entidade ao caminho especificado no campo de URL
- DELETE
  - exclui arquivo especificado no campo de URL

Mensagem de resposta HTTP



Códigos de estado da resposta HTTP

primeira linha da mensagem de resposta servidor->cliente

alguns exemplos de código:

200 OK

- requisição bem-sucedida, objeto requisitado mais adiante

301 Moved Permanently

- objeto requisitado movido, novo local especificado mais adiante na mensagem (Location:)

400 Bad Request

- mensagem de requisição não entendida pelo servidor

404 Not Found

- documento requisitado não localizado neste servidor

505 HTTP Version Not Supported