# Capítulo 2: Camada de aplicação

REDES DE COMPUTADORES E A INTERNET 5ª edição

*Uma Abordagem Top-Down*

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de sockets com UDP
- 2.8 Programação de sockets com TCP

# 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

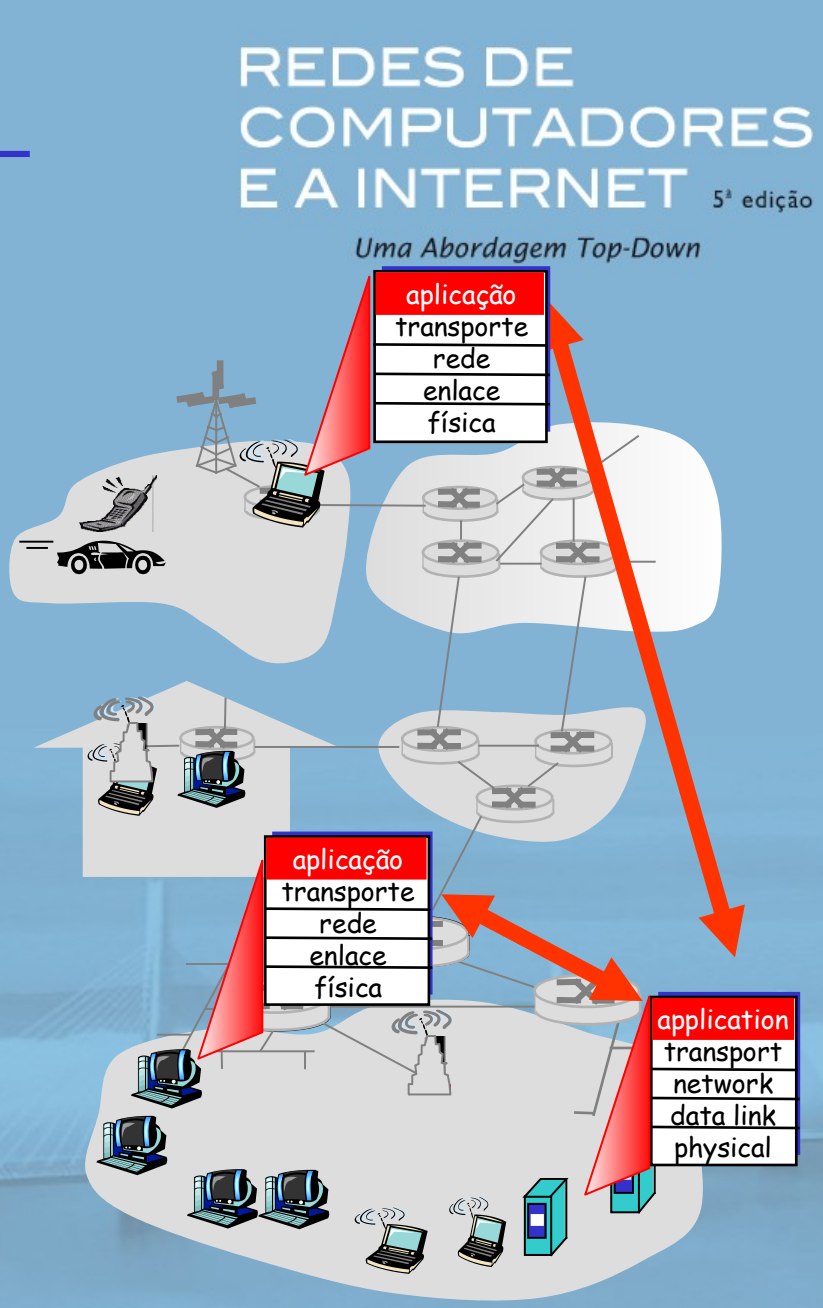
# Criando uma aplicação de rede

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



# Capítulo 2: Camada de aplicação

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

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de sockets com UDP
- 2.8 Programação de sockets com TCP

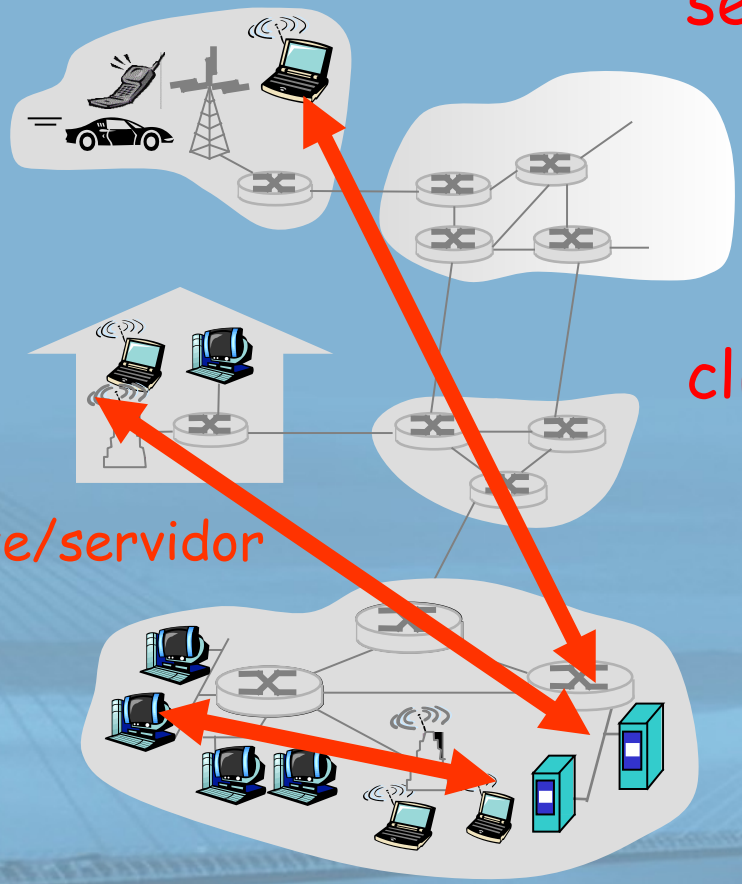
# Arquiteturas de aplicação



- 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

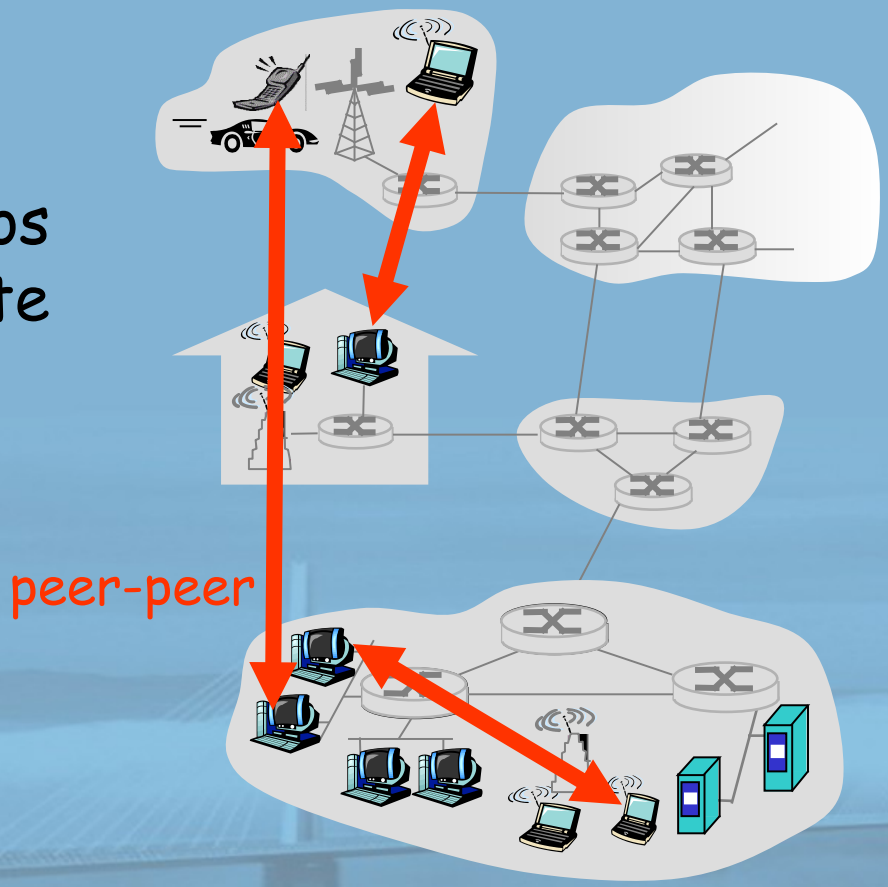
# Centros de dados da Google



- 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

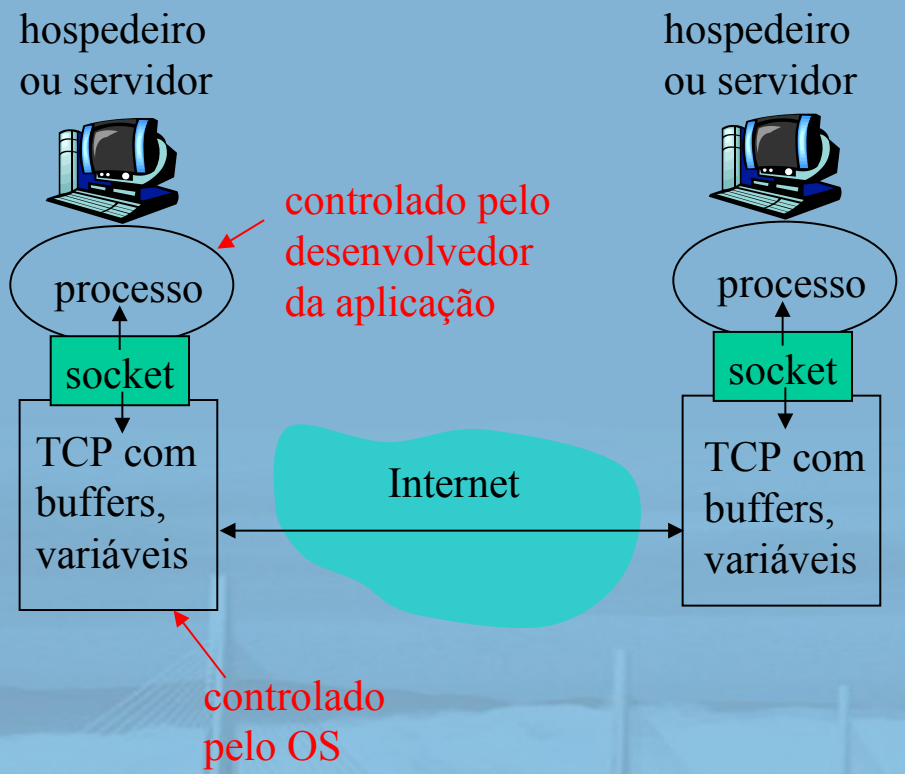
processo cliente: processo que inicia a comunicação

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<br>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 |

# Capítulo 2: Camada de aplicação

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

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de sockets com UDP
- 2.8 Programação de sockets com TCP

# 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`

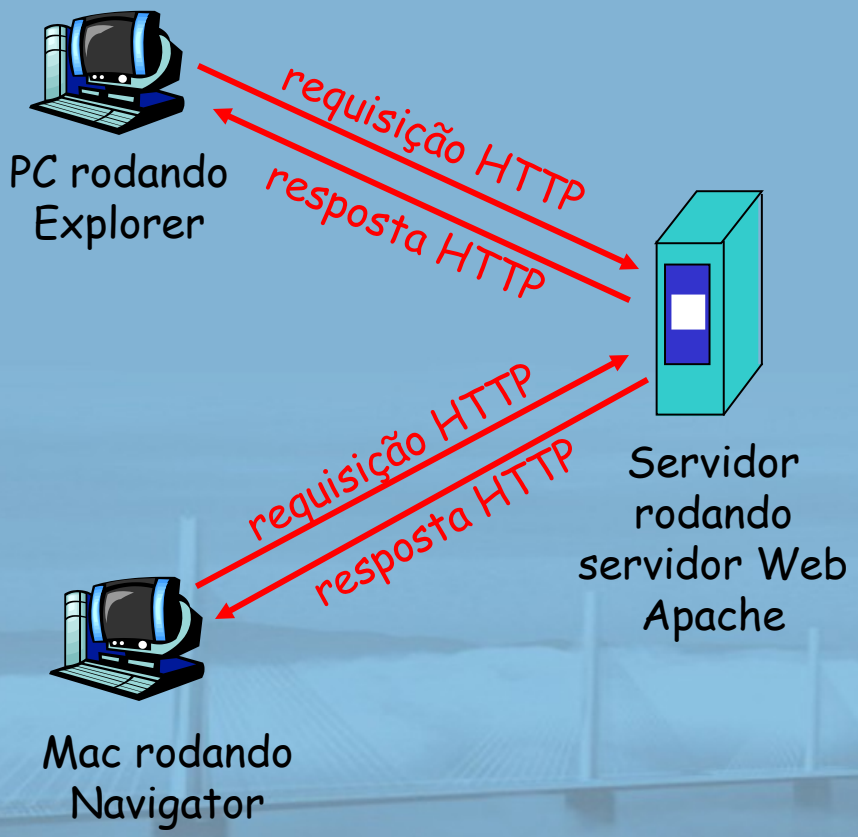
nome do hospedeiro | nome do caminho

# 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)

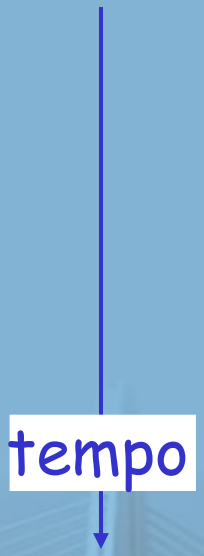
1a. Cliente HTTP inicia conexão TCP com servidor HTTP (processo) em www.someSchool.edu na porta 80.

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

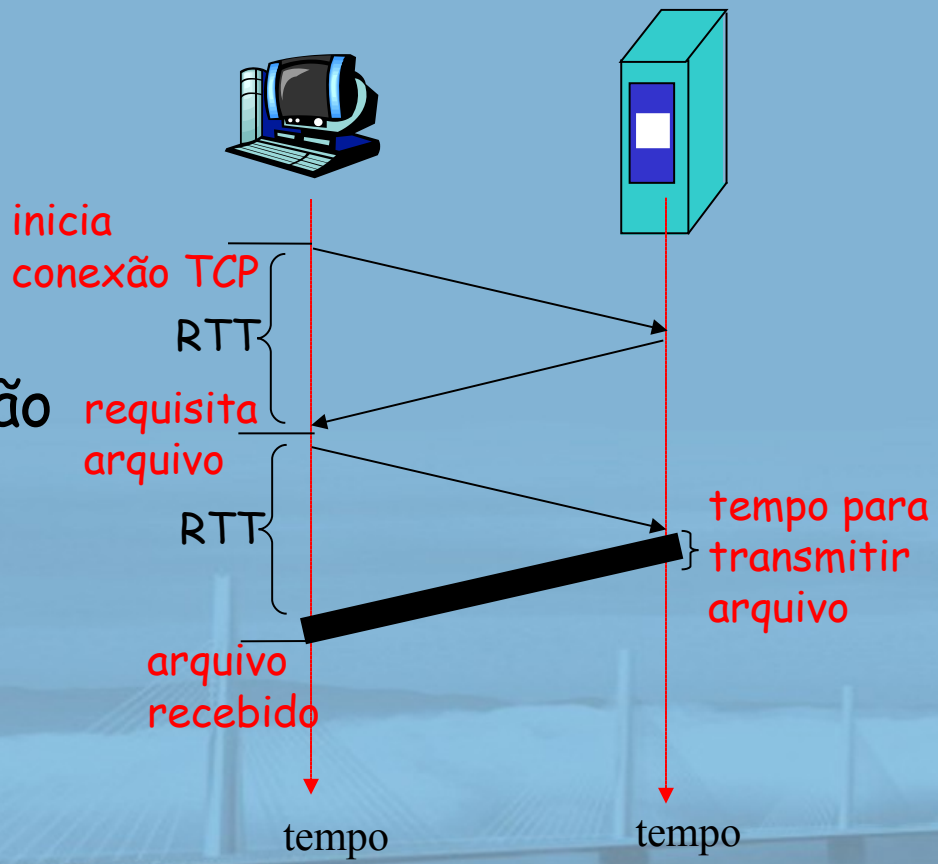


definição de RTT: tempo para um pequeno pacote trafegar do cliente ao servidor e retornar.

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

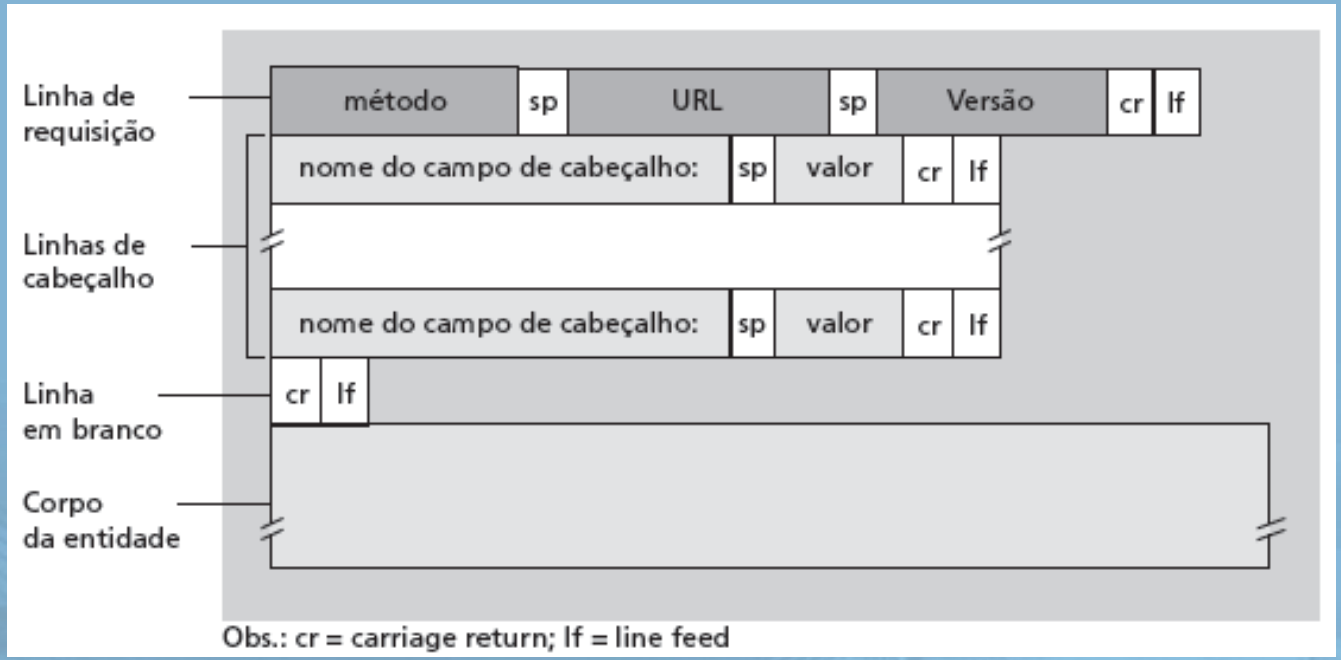
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

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

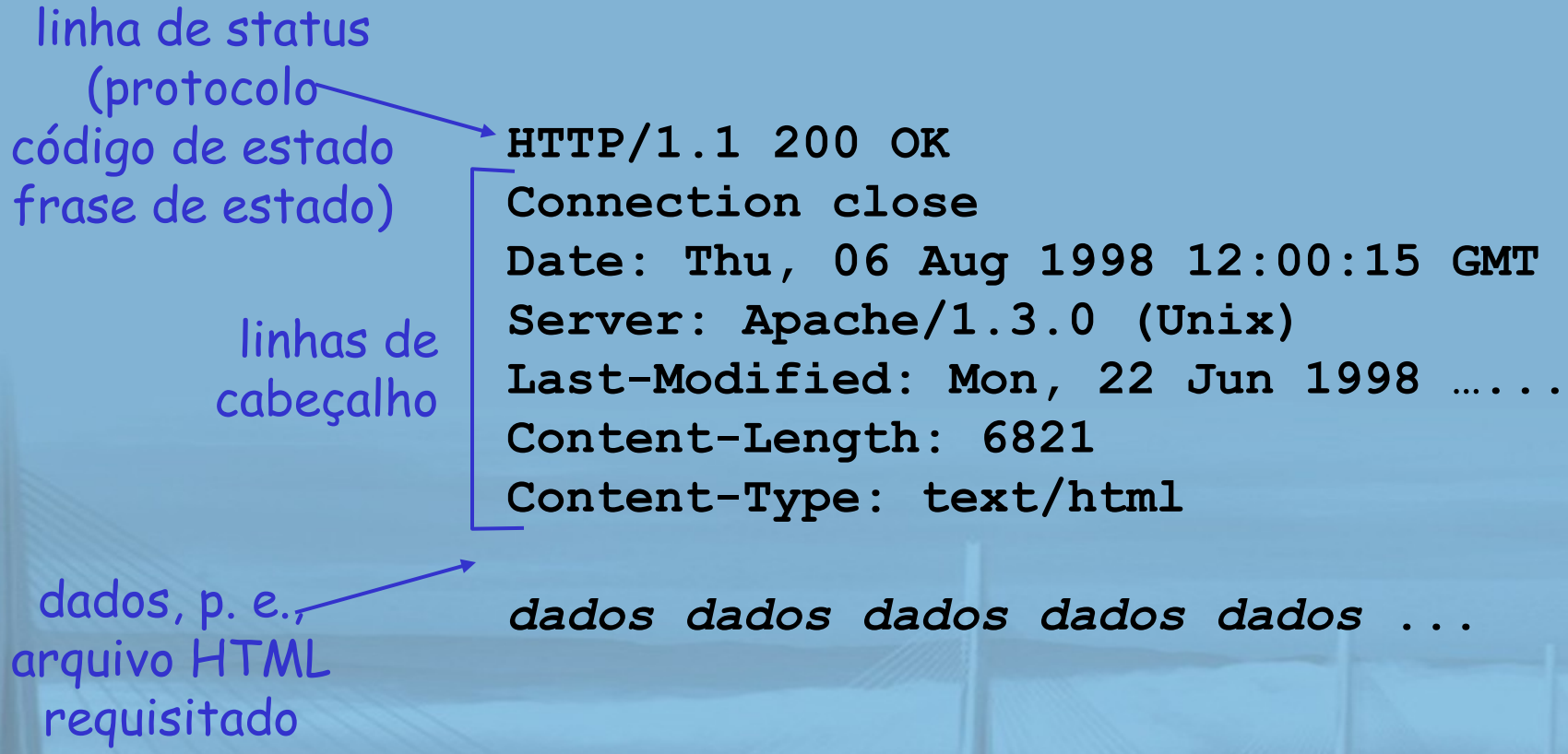
- GET
- POST
- HEAD
  - pede ao servidor para deixar objeto requisitado fora da resposta

## 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**

# Testando o HTTP (lado cliente) você mesmo



1. Use Telnet para seu servidor Web favorito:

```
telnet cis.poly.edu 80
```

Abre conexão TCP com porta 80 (porta HTTP default do servidor) em cis.poly.edu. Qualquer coisa digitada é enviada à porta 80 em cis.poly.edu

2. Digite uma requisição HTTP GET:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

Digitando isto (pressione carriage return duas vezes), você envia esta requisição GET mínima (mas completa) ao servidor HTTP

3. Veja a mensagem de resposta enviada pelo servidor HTTP!

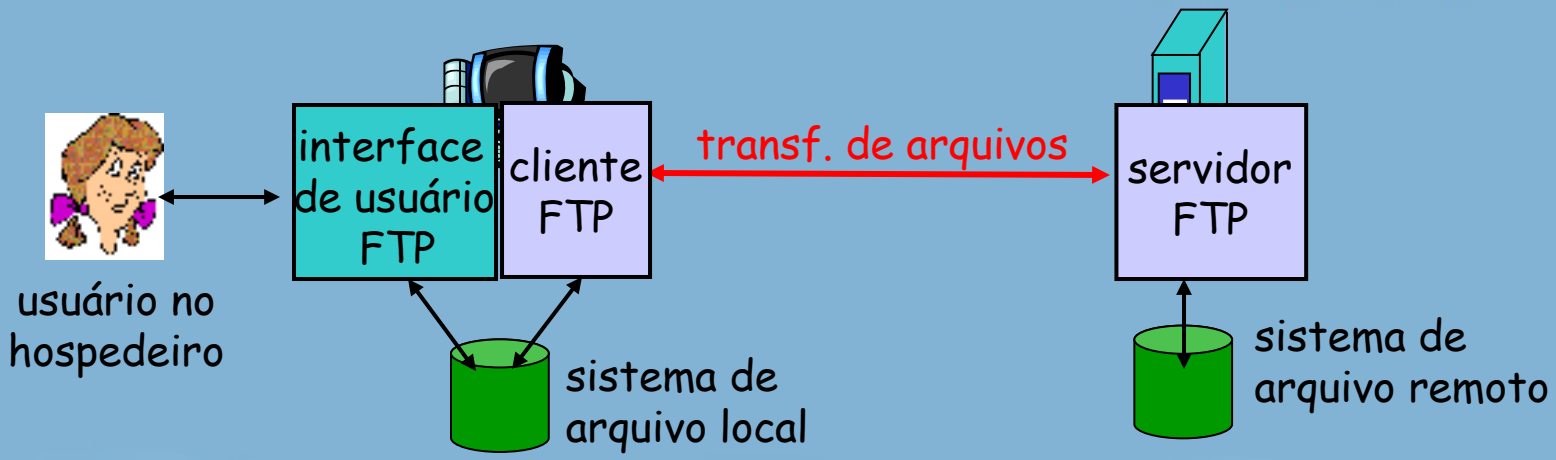
# Capítulo 2: Camada de aplicação

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

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de sockets com UDP
- 2.8 Programação de sockets com TCP

# FTP: o protocolo de transferência de arquivos



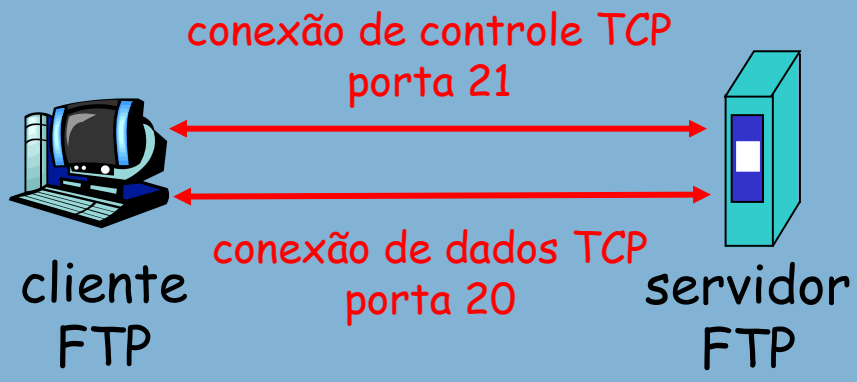



- transfere arquivo de/para hospedeiro remoto
- modelo cliente/servidor
  - *cliente:* lado que inicia transferência (de/para remoto)
  - *servidor:* hospedeiro remoto
- ftp: RFC 959
- servidor ftp: porta 21

# FTP: conexões separadas para controle e dados



- cliente FTP contacta servidor FTP na porta 21, TCP é protocolo de transporte
- cliente autorizado por conexão de controle
- cliente navega por diretório remoto enviando comandos por conexão de controle
- quando servidor recebe comando de transferência de arquivo, abre 2ª conexão TCP (para arquivo) com cliente
- após transferir um arquivo, servidor fecha conexão de dados




- servidor abre outra conexão de dados TCP para transferir outro arquivo
- conexão de controle: "fora da banda"
- servidor FTP mantém "estado": diretório atual, autenticação anterior

# Comandos e respostas FTP



## exemplos de comandos:

- enviado como texto ASCII pelo canal de controle
- **USER** ***nome-usuário***
- **PASS** ***senha***
- **LIST** retorna lista de arquivos no diretório atual
- **RETR** ***nome-arquivo*** recupera (apanha) arquivo
- **STOR** ***nome-arquivo*** armazena (coloca) arquivo no hospedeiro remoto

## exemplos de códigos de retorno

- código e frase de estado (como no HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

# Capítulo 2: Camada de aplicação

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

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de sockets com UDP
- 2.8 Programação de sockets com TCP

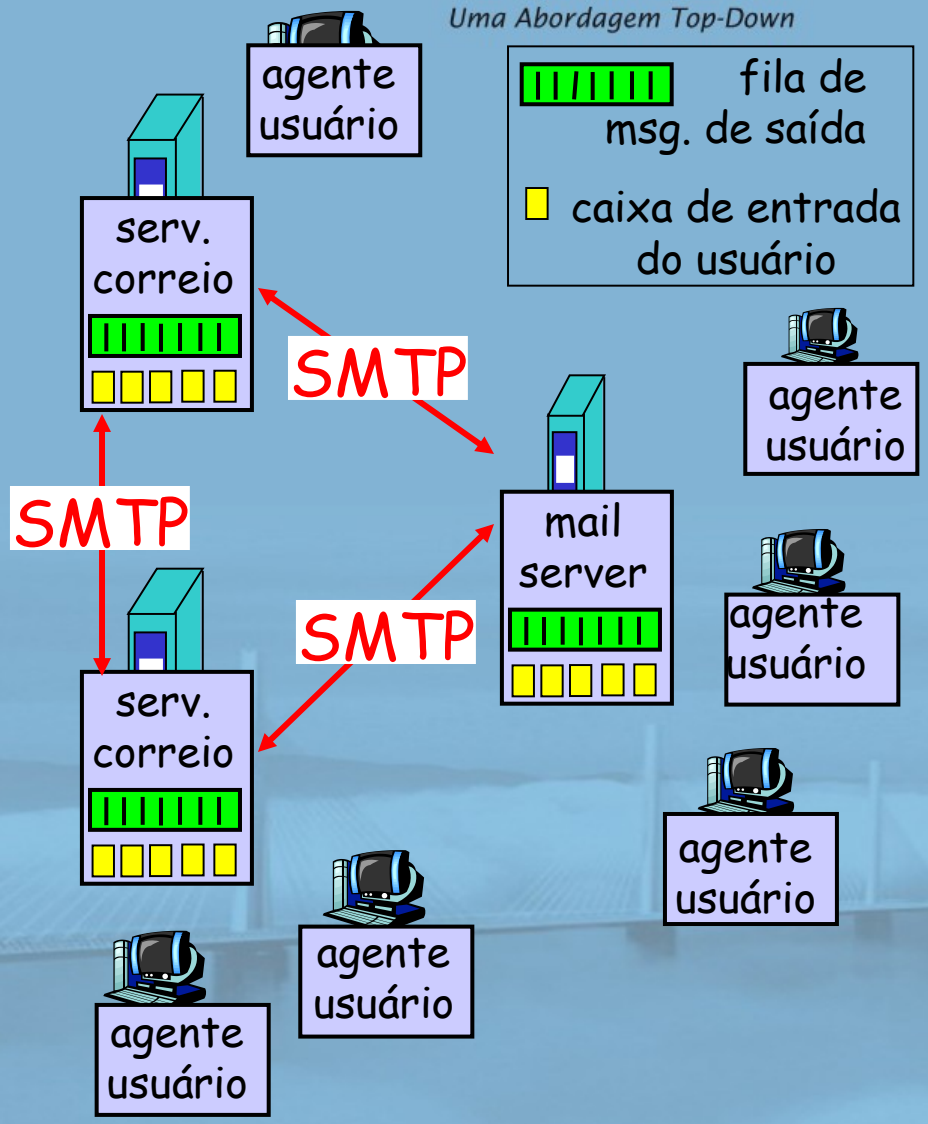
# Correio eletrônico

## Três componentes principais:

- agentes do usuário
- servidores de correio
- Simple Mail Transfer Protocol: SMTP

## Agente do usuário

- também chamado "leitor de correio"
- redigir, editar, ler mensagens de correio eletrônico
- p. e., Eudora, Outlook, elm, Mozilla Thunderbird
- mensagens entrando e saindo armazenadas no servidor

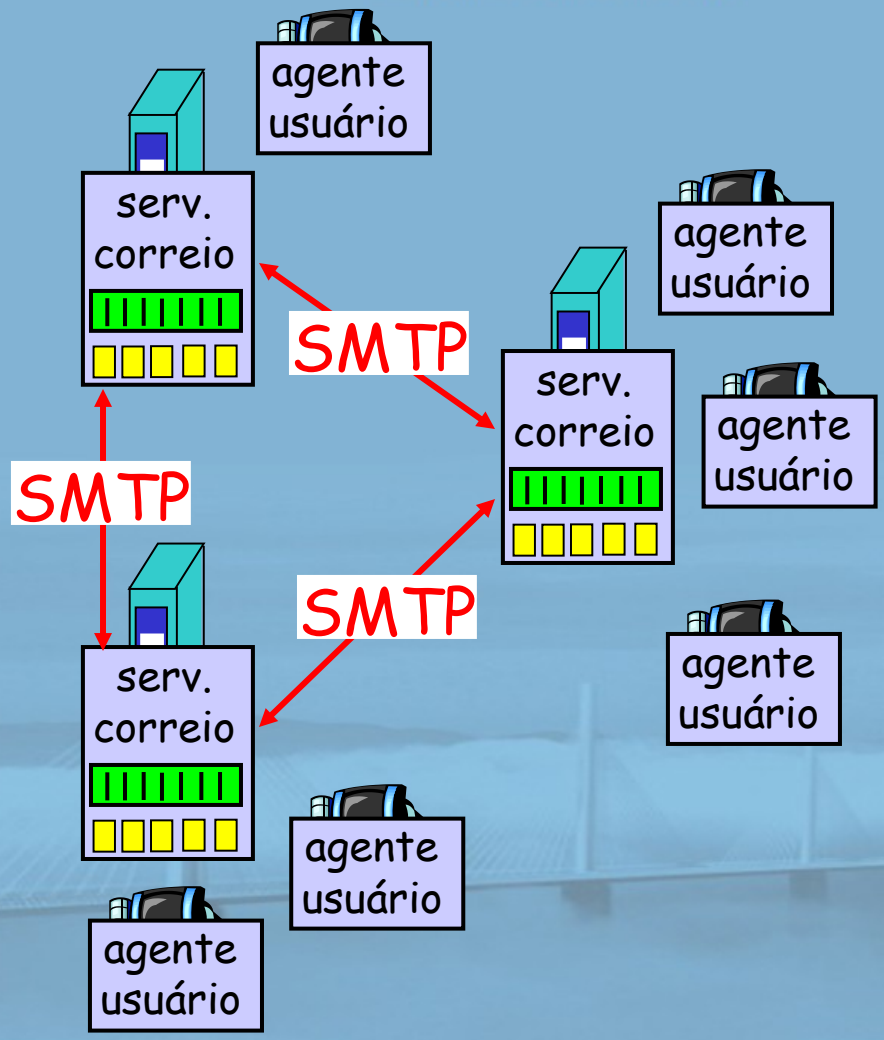


# Correio eletrônico: servidores de correio



## servidores de correio

- caixa de correio contém mensagens que chegam para o usuário
- fila de mensagens com mensagens de correio a serem enviadas
- protocolo SMTP entre servidores de correio para enviar mensagens de e-mail
  - cliente: servidor de envio de correio
  - "servidor": servidor de recepção de correio



# Correio eletrônico: SMTP [RFC 2821]

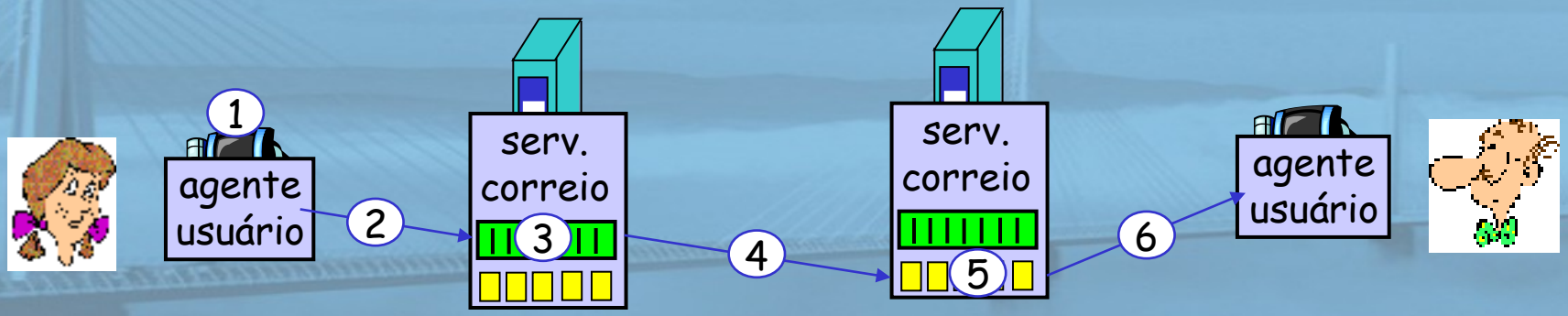


- usa TCP para transferir de modo confiável a mensagem de e-mail do cliente ao servidor, porta 25
- transferência direta: servidor de envio ao servidor de recepção
- três fases da transferência
  - handshaking (saudação)
  - transferência de mensagens
  - fechamento
- interação comando/resposta
  - comandos: texto ASCII
  - resposta: código e frase de estado
- mensagens devem estar em ASCII de 7 bits

# Cenário: Alice envia mensagem a Bob



1) Alice usa AU para redigir mensagem "para" `bob@algumaescola.edu`
2) O AU de Alice envia mensagem ao seu servidor de correio, que é colocada na fila de mensagens
3) Lado cliente do SMTP abre conexão TCP com servidor de correio de Bob
4) Cliente SMTP envia mensagem de Alice pela conexão TCP
5) Servidor de correio de Bob coloca mensagem na caixa de correio de Bob
6) Bob chama seu agente do usuário para ler mensagem



# Exemplo de interação SMTP



```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Você gosta de ketchup?
C: Que tal picles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

# Teste a interação SMTP você mesmo:



- `telnet` ***`nome-servidor`*** `25`
- veja resposta 220 do servidor
- digite comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT

isso permite que você envie e-mail sem usar o cliente de e-mail (leitor)

# SMTP: palavras finais



- SMTP usa conexões persistentes
- SMTP requer que a mensagem (cabeçalho e corpo) esteja em ASCII de 7 bits
- servidor SMTP usa `CRLF.CRLF` para determinar fim da mensagem

## Comparação com HTTP:

- HTTP: puxa
- SMTP: empurra
- ambos têm interação de comando/resposta em ASCII, códigos de estado
- HTTP: cada objeto encapsulado em sua própria mensagem de resposta
- SMTP: múltiplos objetos enviados na mensagem multiparte

# Formato da mensagem de correio

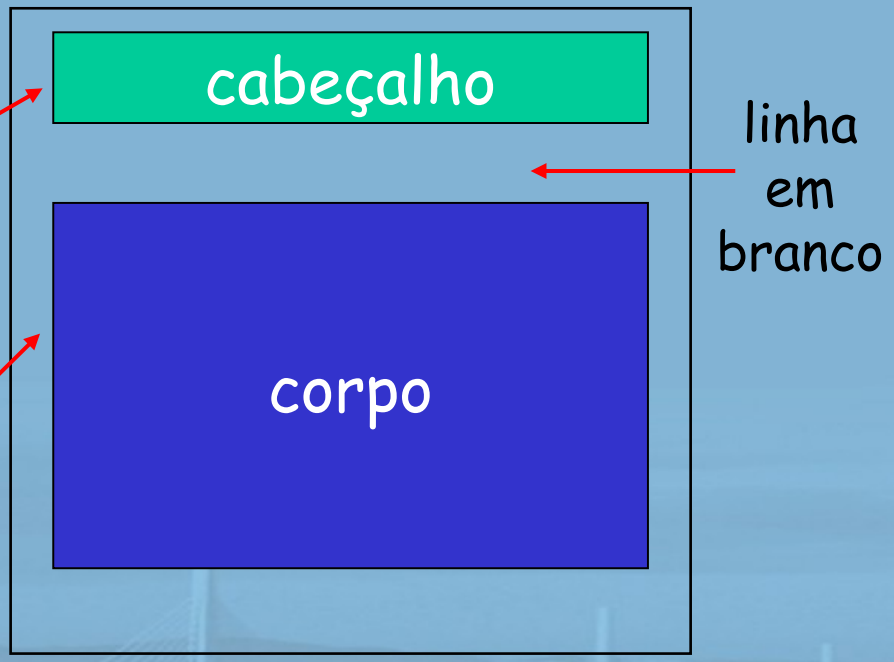


SMTP: protocolo para trocar mensagens de e-mail

RFC 822: padrão para formato de mensagem de texto:

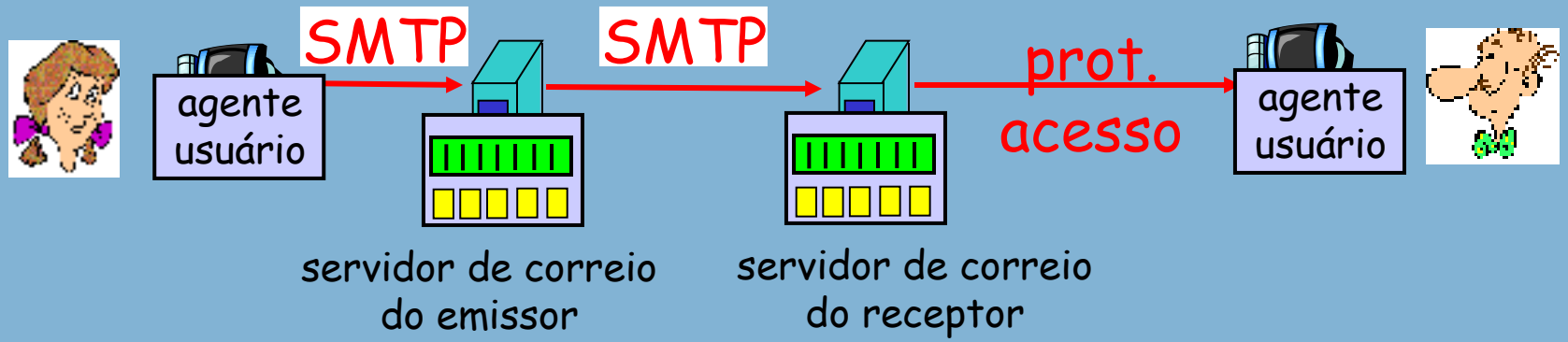
- linhas de cabeçalho, p. e.,
  - Para:
  - De:
  - Assunto:

  *diferente dos comandos SMTP!*
- corpo
  - a "mensagem", apenas em caracteres ASCII



# Protocolos de acesso de correio






- SMTP: remessa/armazenamento no servidor do receptor
- protocolo de acesso ao correio: recuperação do servidor
  - POP: Post Office Protocol [RFC 1939]
    - autorização (agente <--> servidor) e download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - mais recursos (mais complexo)
    - manipulação de msgs armazenadas no servidor
  - HTTP: gmail, Hotmail, Yahoo! Mail etc.

# Protocolo POP3



## fase de autorização

- comandos do cliente:
  - `user:` declare "username"
  - `pass:` senha
- respostas do servidor
  - `+OK`
  - `-ERR`

## fase de transação, cliente:

- `list:` lista números de msg.
- `retr:` recupera mensagem por número
- `dele:` exclui
- `quit`

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK usuário logado com sucesso
```

```
C: list
  S: 1 498
  S: 2 912
  S: .
  C: retr 1
  S: <message 1 contents>
  S: .
  C: dele 1
  C: retr 2
  S: <message 1 contents>
  S: .
  C: dele 2
  C: quit
  S: +OK serv. POP3 desconectando
```

# POP3 (mais) e IMAP

## Mais sobre POP3

- Exemplo anterior usa modo "download e excluir"
- Bob não pode reler e-mail se mudar o cliente
- "Download-e-manter": cópias de mensagens em clientes diferentes
- POP3 é sem estado entre as sessões

## IMAP

- Mantém todas as mensagens em um local: o servidor
- Permite que o usuário organize msgs em pastas
- IMAP mantém estado do usuário entre sessões:
  - nomes de pastas e mapeamento entre IDs de mensagem e nome de pasta