Capítulo 2: Camada de Aplicação

suruagy@cin.ufpe.br

Baseado nos slides de Kurose e Ross

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Correio Eletrônico na Internet
- 2.4 DNS: o serviço de diretório da Internet
- 2.5 Aplicações P2P
- 2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)
- 2.7 Programação de *sockets* com UDP e TCP

Capítulo 2: Camada de Aplicação

Metas do capítulo:

- aspectos conceituais e de implementação de protocolos de aplicação em redes
  - modelos de serviço da camada de transporte
  - paradigma cliente servidor
  - paradigma *peer-to-peer* (p2p)
- aprender sobre protocolos através do estudo de protocolos populares da camada de aplicação:
  - HTTP
  - SMTP/ POP3/ IMAP
  - DNS
- Criar aplicações de rede
  - programação usando a API de *sockets*

Algumas aplicações de rede

- Correio eletrônico
- A Web
- Mensagens instantâneas
- Login em computador remoto como Telnet e SSH
- Compartilhamento de arquivos P2P
- Jogos multiusuários em rede
- *Streaming* de vídeos armazenados (YouTube, Hulu, Netflix)
- Telefonia por IP (Skype)
- Videoconferência em tempo real
- Busca
- ...
- ...

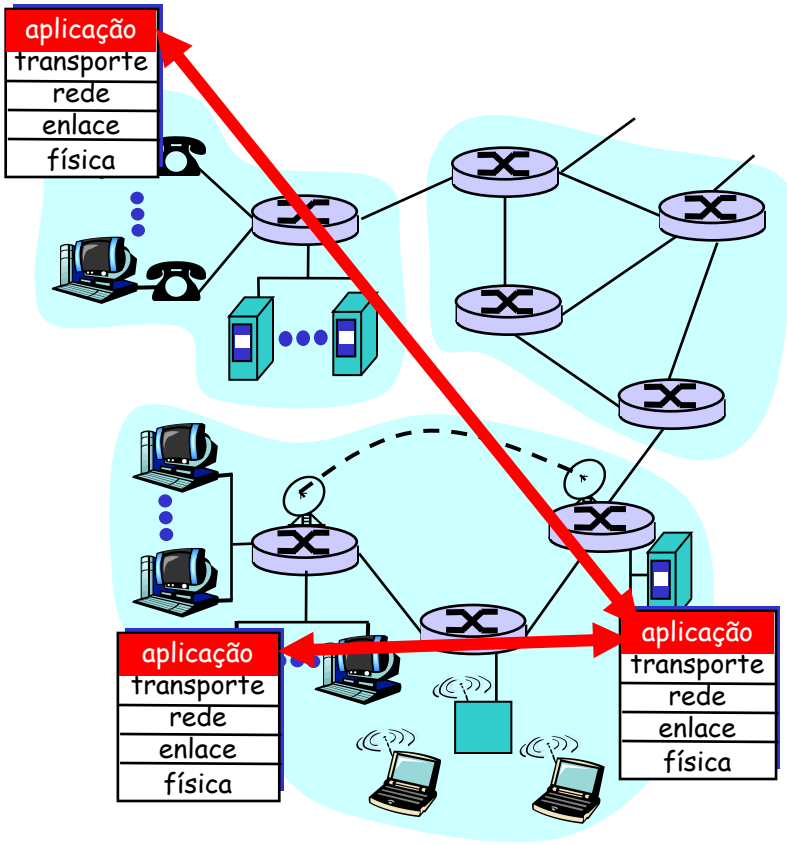
Criando uma aplicação de rede

Programas que

- Executam em (diferentes) sistemas finais
- Comunicam-se através da rede
- p.ex., servidor Web se comunica com o navegador

Programas não relacionados ao núcleo da rede

- Dispositivos do núcleo da rede não executam aplicações dos usuários
- Aplicações nos sistemas finais permitem rápido desenvolvimento e disseminação



Arquiteturas das aplicações de rede

- Estruturas possíveis das aplicações:
  - Cliente-servidor
  - Peer-to-peer (P2P)

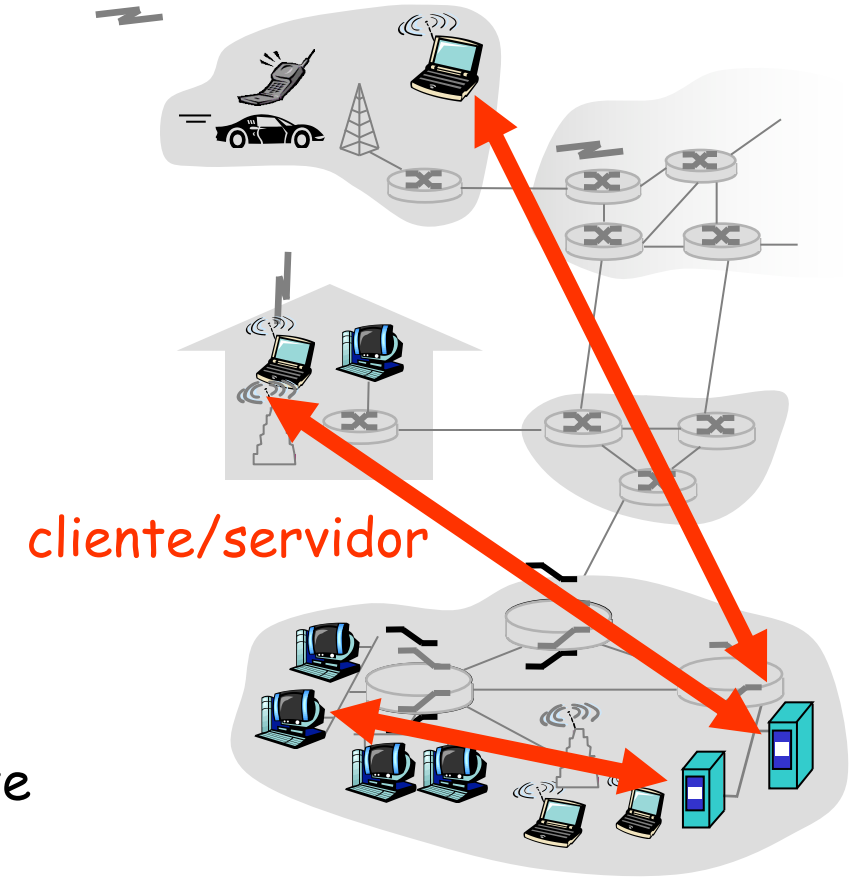
Arquitetura cliente-servidor

Servidor:

- Sempre ligado
- Endereço IP permanente
- Escalabilidade com *data centers*

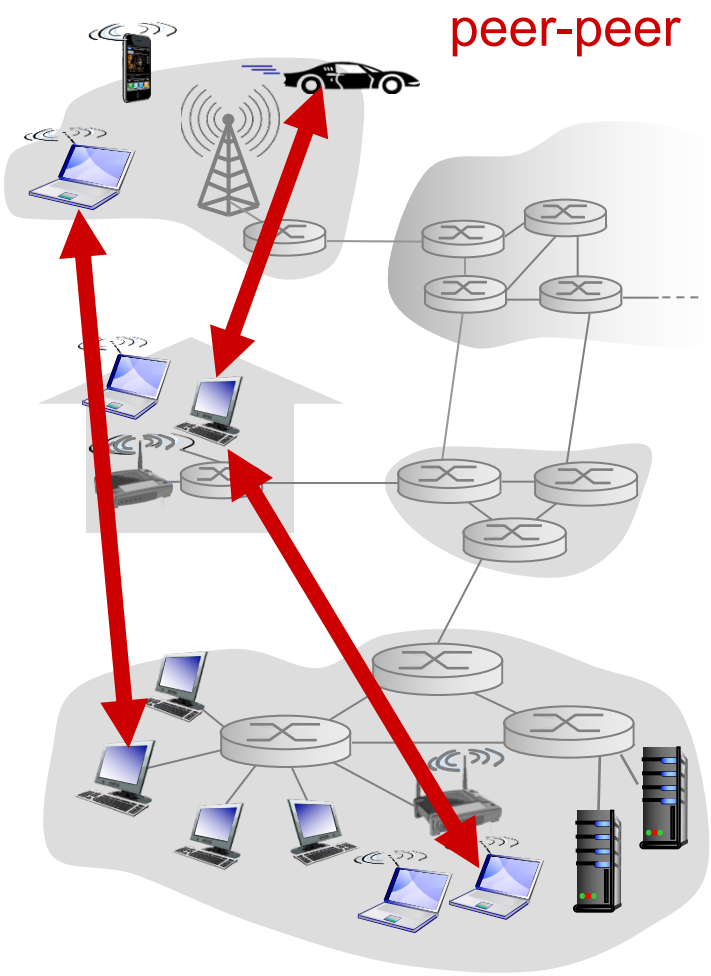
Clientes:

- Comunicam-se com o servidor
- Podem estar conectados intermitentemente
- Podem ter endereços IP dinâmicos
- Não se comunicam diretamente com outros clientes



Arquitetura P2P

- Não há servidor sempre ligado
- Sistemas finais arbitrários se comunicam diretamente
- Pares solicitam serviços de outros pares e em troca proveem serviços para outros parceiros:
  - Autoescalabilidade – novos pares trazem nova capacidade de serviço assim como novas demandas por serviços.
- Pares estão conectados intermitentemente e mudam endereços IP
  - Gerenciamento complexo



Comunicação entre Processos

Processo: programa que executa num sistema final

- processos no mesmo sistema final se comunicam usando comunicação entre processos (definida pelo sistema operacional)
- processos em sistemas finais distintos se comunicam trocando mensagens através da rede

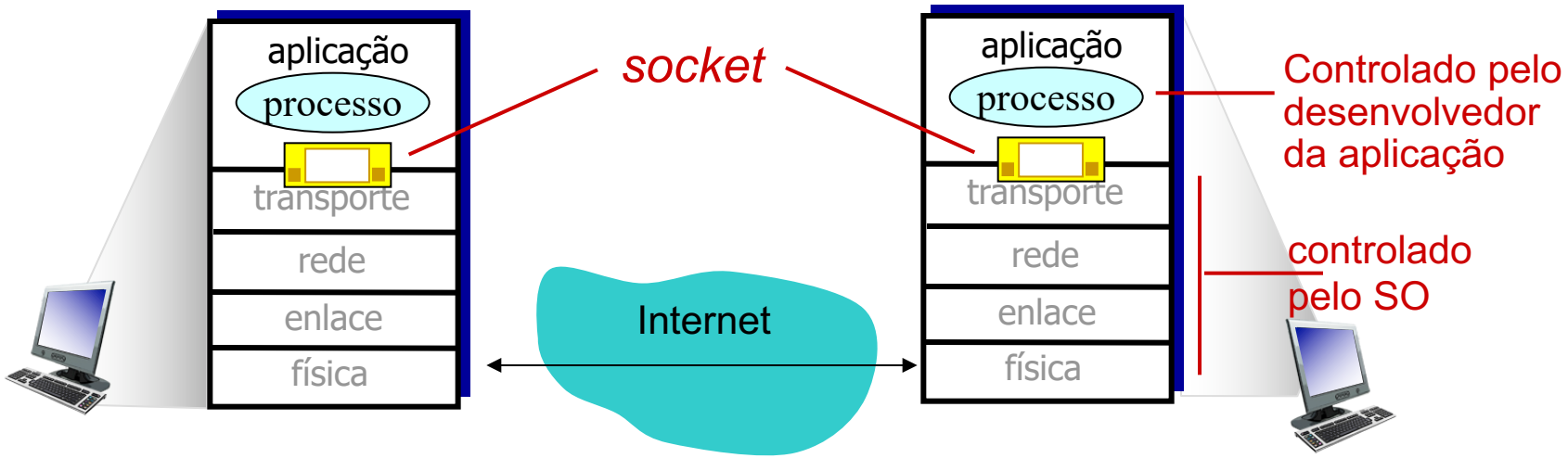
Processo cliente:
processo que inicia a comunicação

Processo servidor:
processo que espera ser contatado

- Nota: aplicações com arquiteturas P2P possuem processos clientes e processos servidores

Sockets

- Os processos enviam/ recebem mensagens para/dos seus *sockets*
- *Um socket* é análogo a uma porta
  - Processo transmissor envia a mensagem através da porta
  - O processo transmissor assume a existência da infraestrutura de transporte no outro lado da porta que faz com que a mensagem chegue ao *socket* do processo receptor



Endereçamento de processos

- Para que um processo receba mensagens, ele deve possuir um identificador
- Cada hospedeiro possui um endereço IP único de 32 bits
- P: o endereço IP do hospedeiro no qual o processo está sendo executado é suficiente para identificar o processo?
- Resposta: Não, muitos processos podem estar executando no mesmo hospedeiro

- O identificador inclui tanto o endereço IP quanto os números das portas associadas com o processo no hospedeiro .
- Exemplo de números de portas:
  - Servidor HTTP: 80
  - Servidor de Correio: 25
- Para enviar uma msg HTTP para o servidor Web gaia.cs.umass.edu
  - Endereço IP: 128.119.245.12
  - Número da porta: 80
- Mais sobre isto posteriormente.

Os protocolos da camada de aplicação definem

- Tipos de mensagens trocadas:
  - ex. mensagens de requisição e resposta
- Sintaxe das mensagens:
  - campos presentes nas mensagens e como são identificados
- Semântica das msgs:
  - significado da informação nos campos
- Regras para quando os processos enviam e respondem às mensagens

Protocolos abertos:

- definidos em RFCs
- Permitem a interoperação
- ex, HTTP e SMTP

Protocolos proprietários:

- Ex., Skype

De que serviços uma aplicação necessita?

Integridade dos dados (sensibilidade a perdas)

- algumas apls (p.ex., transf. de arquivos, transações web) requerem uma transferência 100% confiável
- outras (p.ex. áudio) podem tolerar algumas perdas

Temporização (sensibilidade a atrasos)

- algumas apls (p.ex., telefonia Internet, jogos interativos) requerem baixo retardo para serem "viáveis"

Vazão (*throughput*)

- algumas apls (p.ex., multimídia) requerem quantia mínima de vazão para serem "viáveis"
- outras apls ("apls elásticas") conseguem usar qq quantia de banda disponível

Segurança

- Criptografia, integridade dos dados, ...

Requisitos de aplicações de rede selecionadas

Aplicação	Sensib. a Perdas	Vazão	Sensibilidade a atrasos
transferência de arqs	sem perdas	elástica	não
correio	sem perdas	elástica	não
documentos Web	sem perdas	elástica	não
áudio/vídeo em tempo real	tolerante	áudio: 5kbps-1Mbps vídeo:10kbps-5Mbps	sim, 100's mseg
áudio/vídeo gravado	tolerante	Igual acima	sim, alguns segs
jogos interativos	tolerante	Alguns kbps-10Mbps	sim, 100's mseg
mensagem instantânea	sem perdas	elástica	sim e não

Serviços providos pelos protocolos de transporte da Internet

Serviço TCP:

- *transporte confiável* entre processos remetente e receptor
- *controle de fluxo:* remetente não vai "afogar" receptor
- *controle de congestionamento:* estrangular remetente quando a rede estiver carregada
- *não provê:* garantias temporais ou de banda mínima
- *orientado a conexão:* apresentação requerida entre cliente e servidor

Serviço UDP:

- transferência de dados não confiável entre processos remetente e receptor
- não provê: estabelecimento da conexão, confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais ou de banda mínima

P: Qual é o interesse em ter um protocolo como o UDP?

Apls Internet: seus protocolos e seus protocolos de transporte

Aplicação	Protocolo da camada de apl.	Protocolo de transporte usado
correio eletrônico	SMTP [RFC 2821]	TCP
acesso terminal remoto	telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transferência de arquivos	FTP [RFC 959]	TCP
streaming multimídia	HTTP (ex. Youtube) RTP [RFC 1889]	TCP ou UDP
telefonia Internet	SIP, RTP, proprietário (ex., Skype)	TCP ou UDP

Tornando o TCP seguro

TCP & UDP

- Sem criptografia
- Senhas em texto aberto enviadas aos sockets atravessam a Internet em texto aberto

SSL

- Provê conexão TCP criptografada
- Integridade dos dados
- Autenticação do ponto terminal

SSL está na camada de aplicação

- Aplicações usam bibliotecas SSL, que "falam" com o TCP

API do socket SSL

- Senhas em texto aberto enviadas ao socket atravessam a rede criptografadas
- Vide Capítulo 7

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Correio Eletrônico na Internet
- 2.4 DNS: o serviço de diretório da Internet
- 2.5 Aplicações P2P
- 2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)
- 2.7 Programação de *sockets* com UDP e TCP

A Web e o HTTP

Primeiro uma revisão...

- Páginas Web consistem de objetos
- um objeto pode ser um arquivo HTML, uma imagem JPEG, um applet Java, um arquivo de áudio,...
- Páginas Web consistem de um arquivo base HTML que inclui vários objetos referenciados
- Cada objeto é endereçável por uma URL
- Exemplo de URL:

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

nome do hospedeiro | nome do caminho

Protocolo HTTP

HTTP: *hypertext transfer protocol*

- protocolo da camada de aplicação da Web
- modelo cliente/servidor
  - *cliente: browser* que pede, recebe (usando o protocolo HTTP) e "visualiza" objetos Web
  - *servidor:* servidor Web envia (usando o protocolo HTTP) objetos em resposta a pedidos

Mais sobre o protocolo HTTP

Usa serviço de transporte TCP:

- cliente inicia conexão TCP (cria *socket*) ao servidor, porta 80
- servidor aceita conexão TCP do cliente
- mensagens HTTP (mensagens do protocolo da camada de apl) trocadas entre *browser* (cliente HTTP) e servidor Web (servidor HTTP)
- encerra conexão TCP

HTTP é "sem estado"

- servidor não mantém informação sobre pedidos anteriores do cliente

Nota

Protocolos que mantêm "estado" são complexos!

- história passada (estado) tem que ser guardada
- Caso caia servidor/cliente, suas visões do "estado" podem ser inconsistentes, devem ser reconciliadas

Conexões HTTP

HTTP não persistente

- No máximo um objeto é enviado numa conexão TCP
  - A conexão é então encerrada
- Baixar múltiplos objetos requer o uso de múltiplas conexões

HTTP persistente

- Múltiplos objetos podem ser enviados sobre uma única conexão TCP entre cliente e servidor

Exemplo de HTTP não persistente

Supomos que usuário digita a URL
www.algumaUniv.br/algumDepartmento/inicial.index

(contém texto, referências a 10 imagens jpeg)

1a. Cliente http inicia conexão TCP a servidor http (processo) a www.algumaUniv.br. Porta 80 é padrão para servidor http.

1b. servidor http no hospedeiro www.algumaUniv.br espera por conexão TCP na porta 80. "aceita" conexão, avisando ao cliente

2. cliente http envia *mensagem de pedido* de http (contendo URL) através do socket da conexão TCP. A mensagem indica que o cliente deseja receber o objeto algumDepartamento/inicial.index

3. servidor http recebe mensagem de pedido, formula *mensagem de resposta* contendo objeto solicitado e envia a mensagem via socket

tempo

Exemplo de HTTP não persistente (cont.)

4. servidor http encerra conexão TCP .

5. cliente http recebe mensagem de resposta contendo arquivo html, visualiza html. Analisando arquivo html, encontra 10 objetos jpeg referenciados

6. Passos 1 a 5 repetidos para cada um dos 10 objetos jpeg

tempo

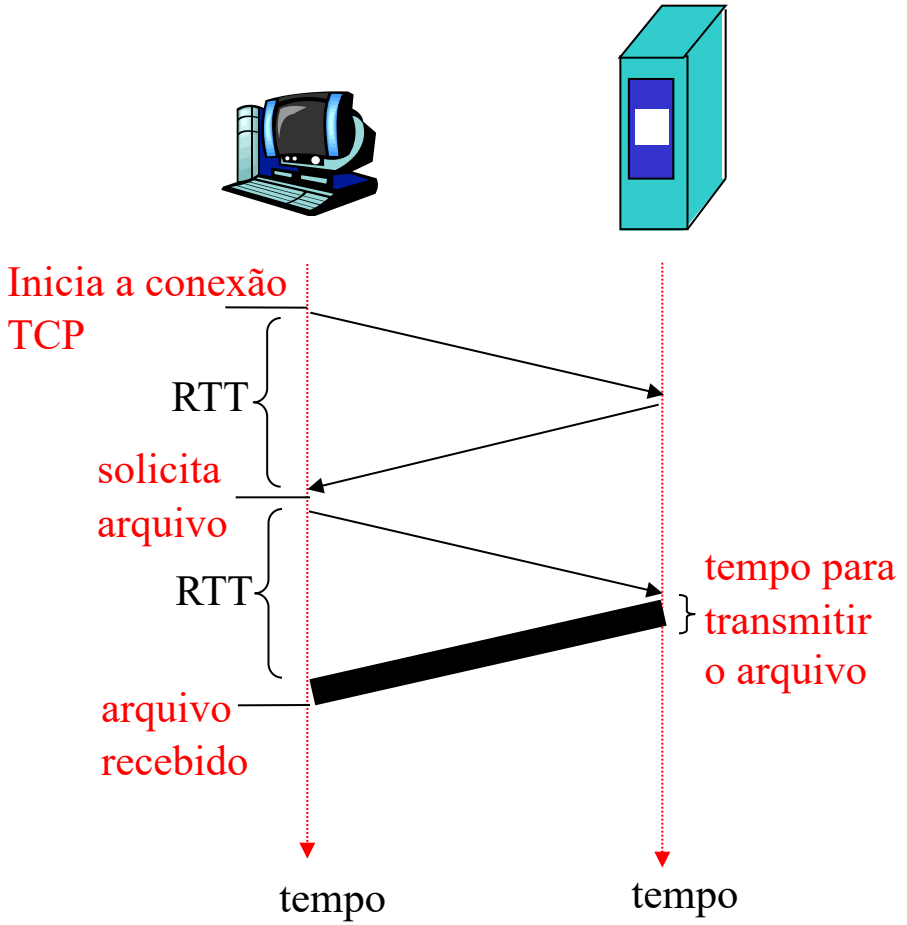
Modelagem do tempo de resposta

Definição de RTT *(Round Trip Time)*: intervalo de tempo entre a ida e a volta de um pequeno pacote entre um cliente e um servidor

Tempo de resposta:

- um RTT para iniciar a conexão TCP
- um RTT para o pedido HTTP e o retorno dos primeiros bytes da resposta HTTP
- tempo de transmissão do arquivo

total = 2RTT+tempo de transmissão do arquivo



HTTP persistente

Problemas com o HTTP não persistente:

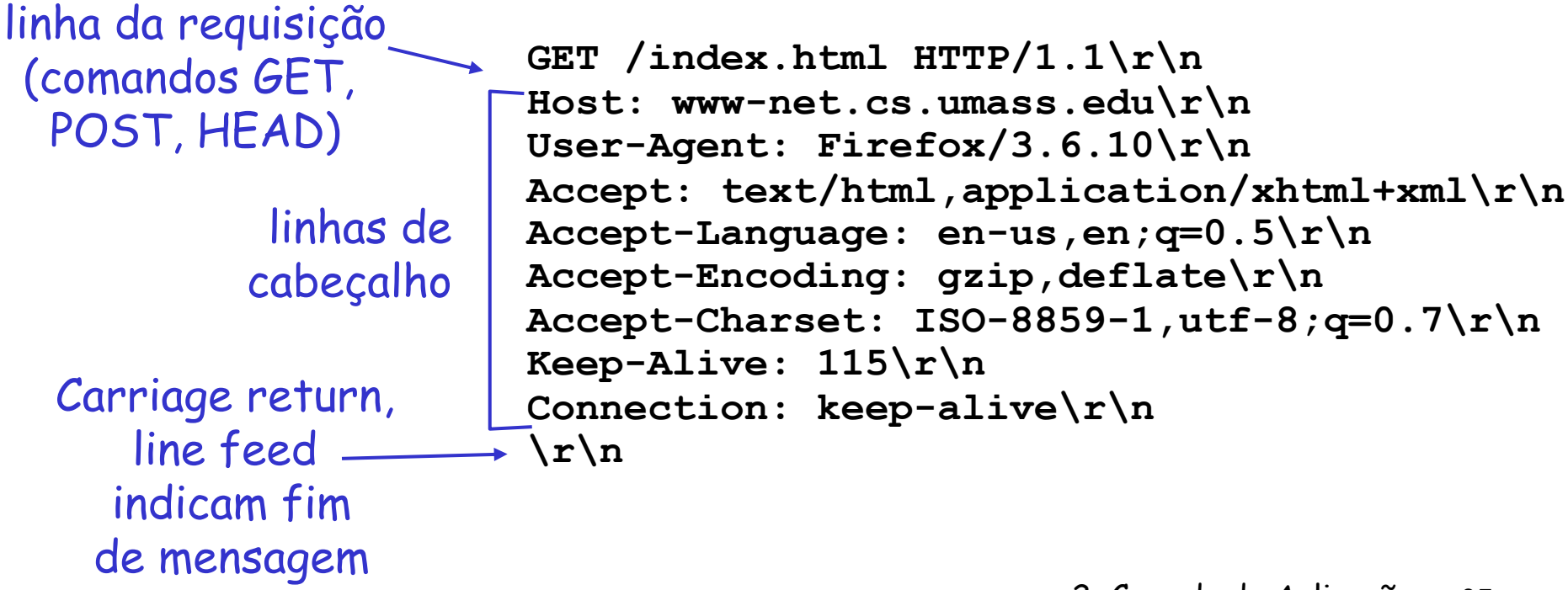
- requer 2 RTTs para cada objeto
- SO aloca recursos do hospedeiro (*overhead*) para cada conexão TCP
- os *browser* frequentemente abrem conexões TCP paralelas para recuperar os objetos referenciados

HTTP persistente

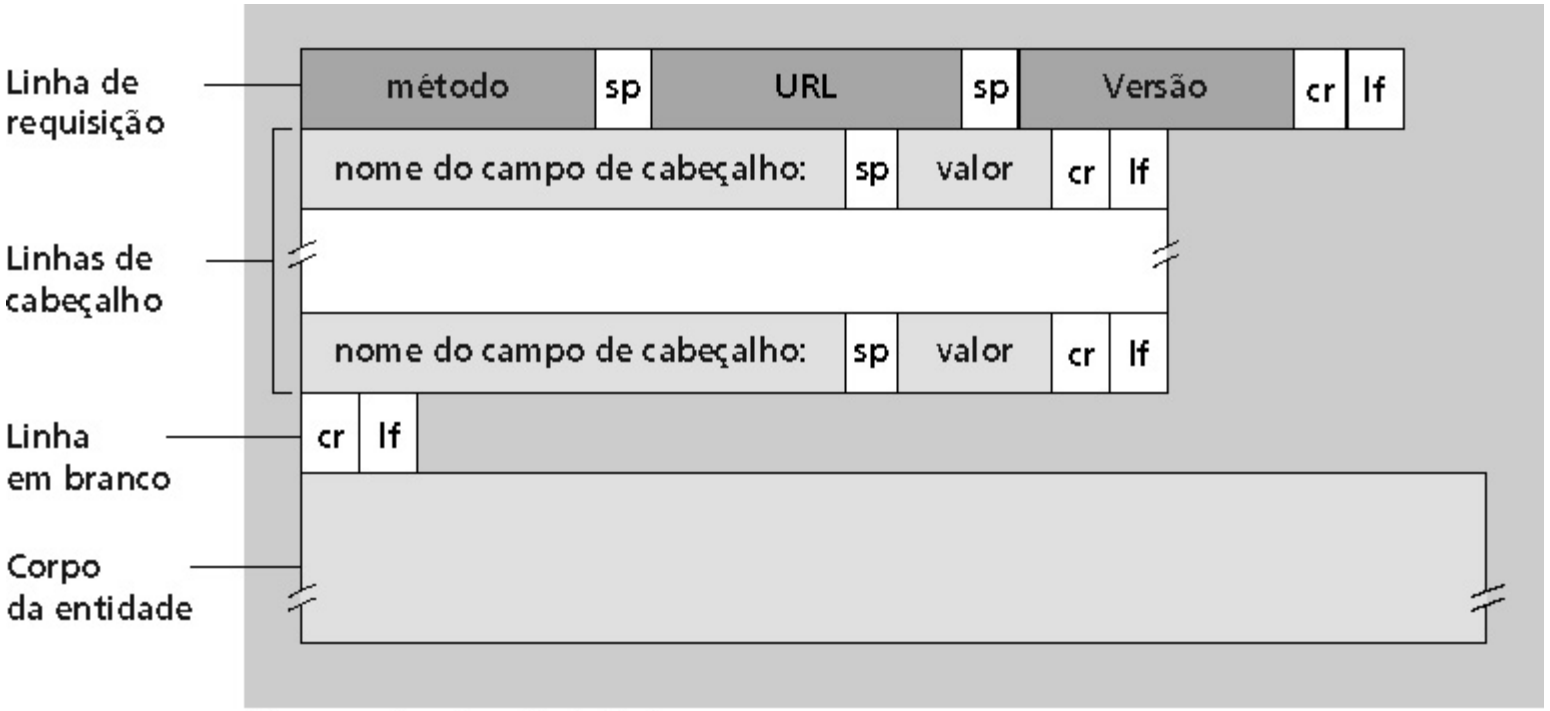
- o servidor deixa a conexão aberta após enviar a resposta
- mensagens HTTP seguintes entre o mesmo cliente/servidor são enviadas nesta conexão aberta
- o cliente envia os pedidos logo que encontra um objeto referenciado
- pode ser necessário apenas um RTT para todos os objetos referenciados

Mensagem de requisição HTTP

- Dois tipos de mensagem HTTP: *requisição, resposta*
- mensagem de requisição HTTP:
  - ASCII (formato legível por pessoas)



Mensagem de requisição HTTP: formato geral



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

Enviando conteúdo de formulário

Método POST :

- Páginas Web frequentemente contêm formulário de entrada
- Conteúdo é enviado para o servidor no corpo da mensagem

Método URL:

- Usa o método GET
- Conteúdo é enviado para o servidor no campo URL:

```
www.somesite.com/animalsearch?key=monkeys&bananas
```

Tipos de métodos

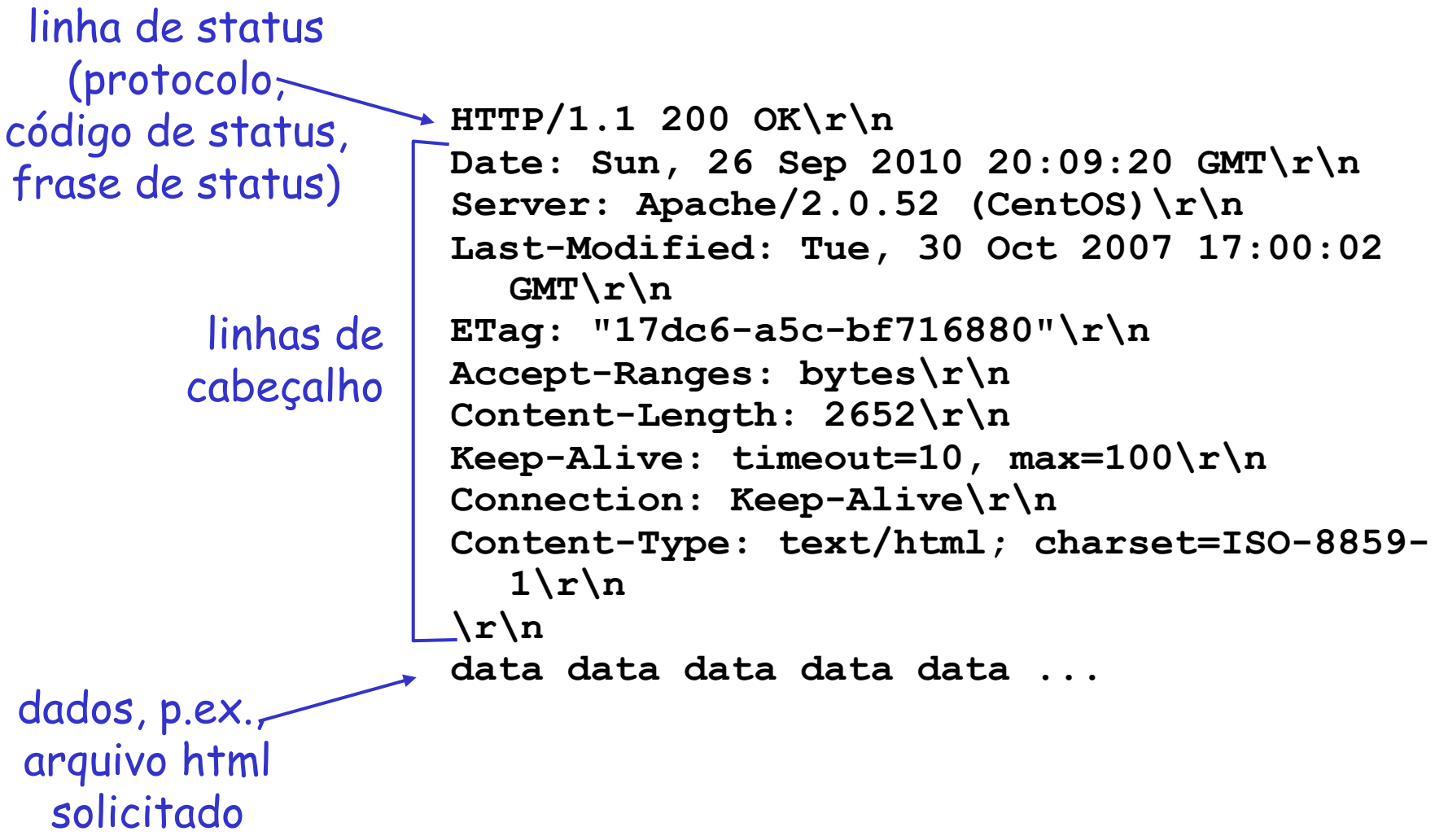
HTTP/1.0

- GET
- POST
- HEAD
  - Pede para o servidor não enviar o objeto requerido junto com a resposta

HTTP/1.1

- GET, POST, HEAD
- PUT
  - *Upload* de arquivo contido no corpo da mensagem para o caminho especificado no campo URL
- DELETE
  - Exclui arquivo especificado no campo URL

Mensagem de resposta HTTP



códigos de status da resposta HTTP

Na primeira linha da mensagem de resposta servidor->cliente. Alguns códigos típicos:

200 OK

- sucesso, objeto pedido segue mais adiante nesta mensagem

301 Moved Permanently

- objeto pedido mudou de lugar, nova localização especificado mais adiante nesta mensagem (Location:)

400 Bad Request

- mensagem de pedido não entendida pelo servidor

404 Not Found

- documento pedido não se encontra neste servidor

505 HTTP Version Not Supported

- versão de http do pedido não usada por este servidor

Experimente você com HTTP (do lado cliente)

1. Use cliente telnet para seu servidor WWW favorito:

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

Abre conexão TCP para a porta 80 (porta padrão do servidor http) a cis.poly.edu. Qualquer coisa digitada é enviada para a porta 80 do cis.poly.edu

2. Digite um pedido GET HTTP:

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

Digitando isto (deve teclar ENTER duas vezes), está enviando este pedido GET mínimo (porém completo) ao servidor http

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

(ou use Wireshark para ver as msgs de pedido/resposta HTTP capturadas)

Cookies: manutenção do "estado" da conexão

Muitos dos principais sítios Web usam *cookies*

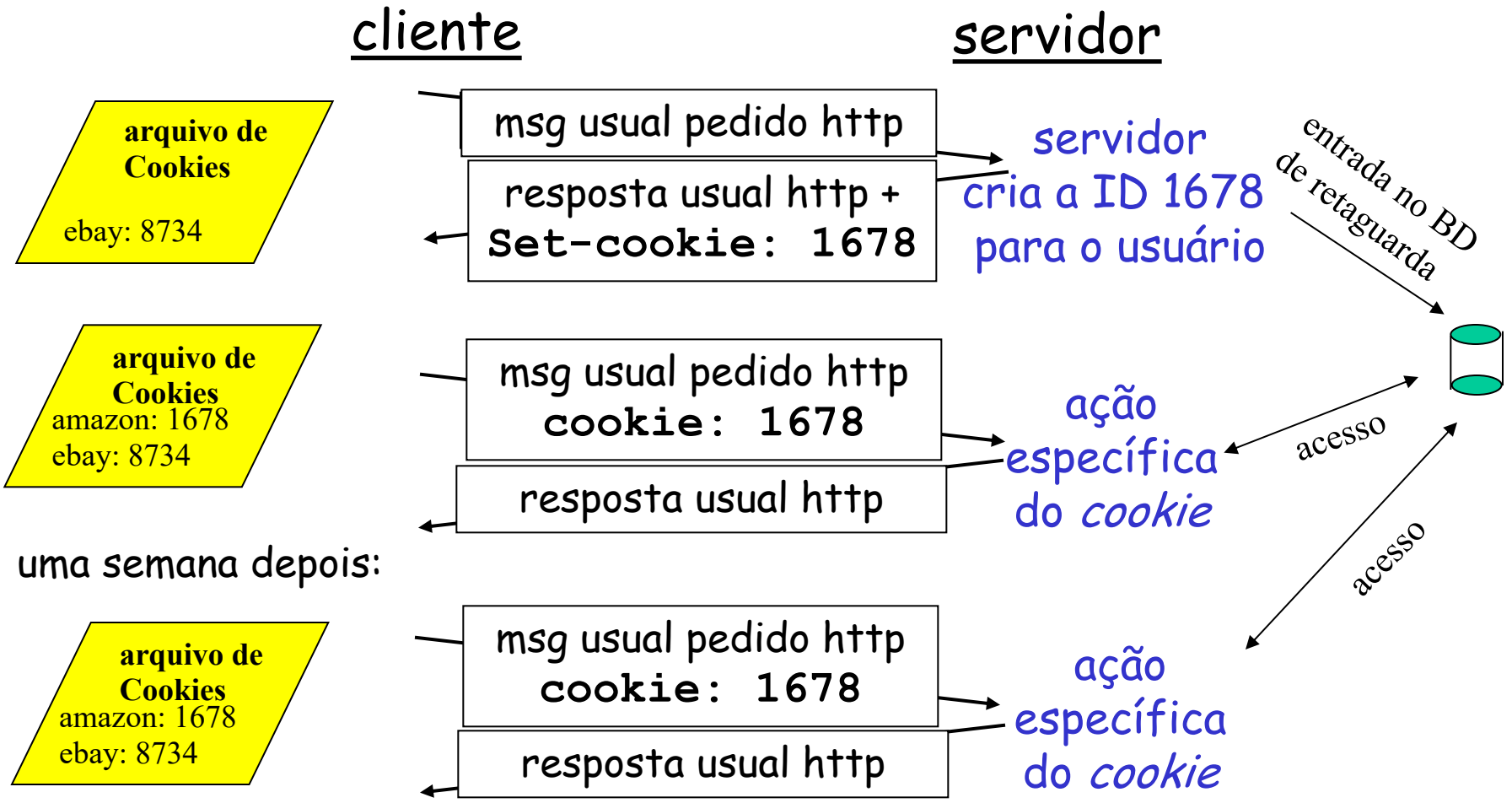
Quatro componentes:

1) linha de cabeçalho do *cookie* na mensagem de resposta HTTP
2) linha de cabeçalho do *cookie* na mensagem de pedido HTTP
3) arquivo do *cookie* mantido no host do usuário e gerenciado pelo browser do usuário
4) BD de retaguarda no sítio Web

Exemplo:

- Suzana acessa a Internet sempre do mesmo PC
- Ela visita um sítio específico de comércio eletrônico pela primeira vez
- Quando os pedidos iniciais HTTP chegam no sítio, o sítio cria
  - uma ID única
  - uma entrada para a ID no BD de retaguarda

Cookies: manutenção do "estado" (cont.)



Cookies (continuação)

O que os *cookies* podem obter:

- autorização
- carrinhos de compra
- recomendações
- estado da sessão do usuário (*Webmail*)

nota

Cookies e privacidade:

- cookies permitem que os sítios aprendam muito sobre você
- você pode fornecer nome e e-mail para os sítios

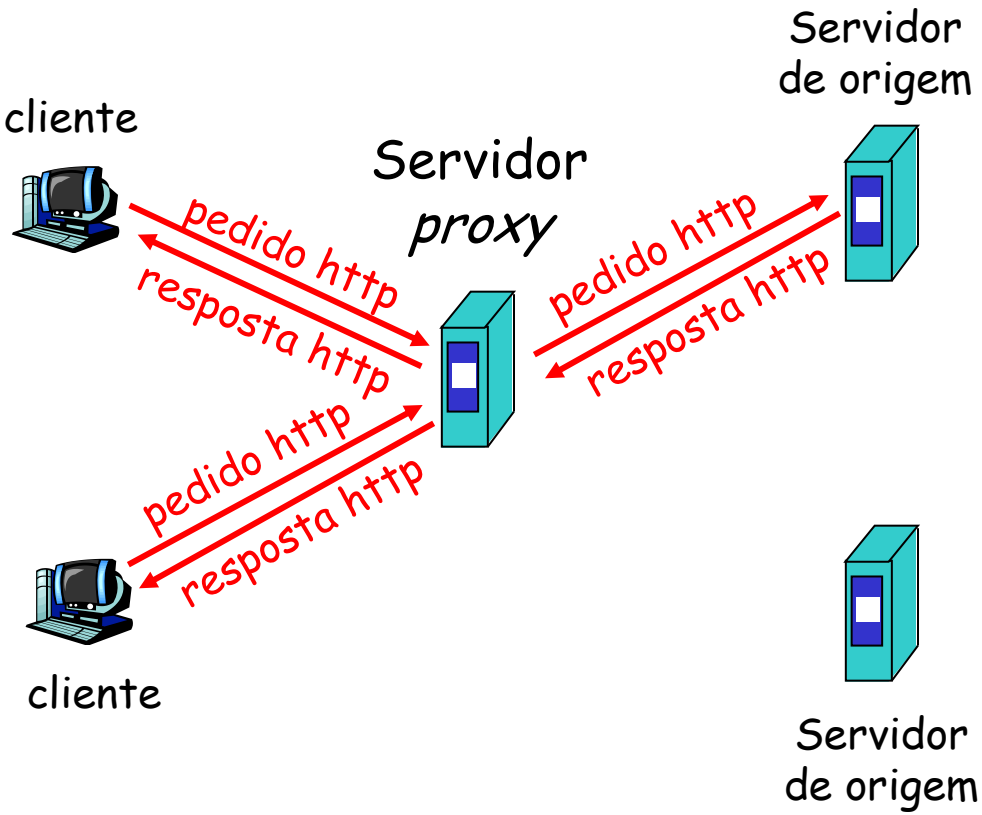
Como manter o "estado":

- Pontos finais do protocolo: mantêm o estado no transmissor/receptor para múltiplas transações
- *Cookies:* mensagens http transportam o estado

Cache Web (servidor proxy)

Meta: atender pedido do cliente sem envolver servidor de origem

- usuário configura *browser*: acessos Web via proxy
- cliente envia todos pedidos HTTP ao *proxy*
  - se objeto estiver no cache do *proxy*, este o devolve imediatamente na resposta HTTP
  - senão, solicita objeto do servidor de origem, depois devolve resposta HTTP ao cliente



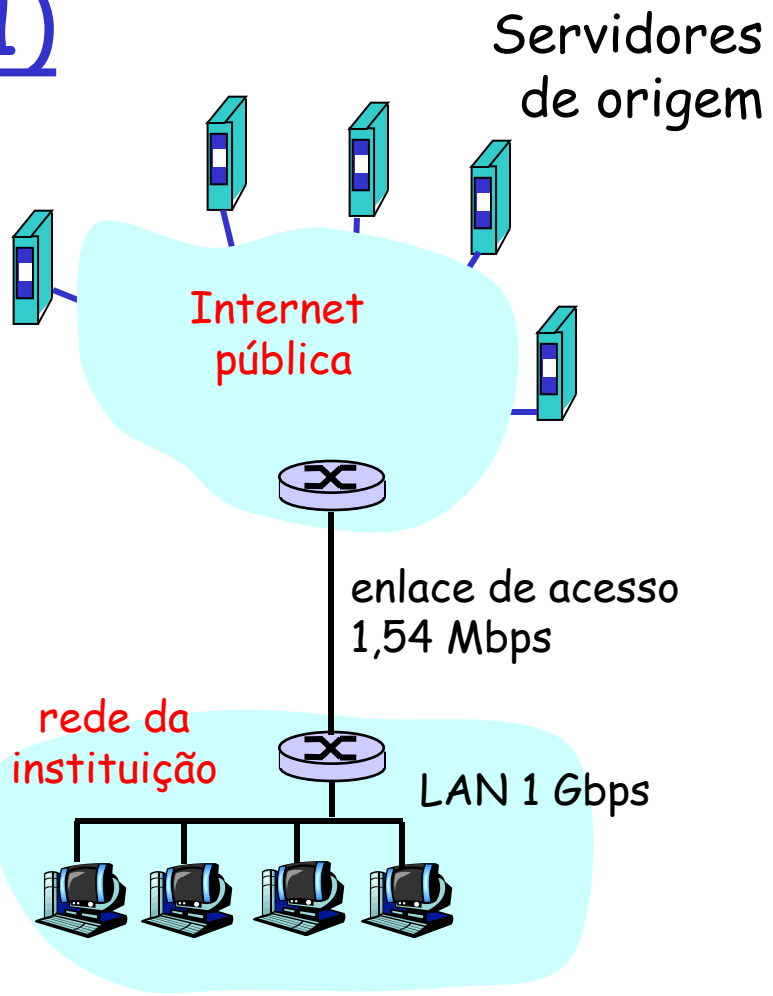
Mais sobre Caches Web

- Cache atua tanto como cliente quanto como servidor
- Tipicamente o cache é instalado por um ISP (universidade, empresa, ISP residencial)

Para que fazer cache Web?

- Redução do tempo de resposta para os pedidos do cliente
- Redução do tráfego no canal de acesso de uma instituição
- A Internet cheia de caches permitem que provedores de conteúdo "pobres" efetivamente forneçam conteúdo (mas o compartilhamento de arquivos P2P também!)

Exemplo de cache (1)



Hipóteses

- Tamanho médio de um objeto = 100.000 bits
- Taxa média de solicitações dos *browsers* de uma instituição para os servidores originais = 15/seg
- Atraso do roteador institucional para qualquer servidor origem e de volta ao roteador = 2seg

Consequências

- Utilização da LAN = 0,15%
- Utilização do canal de acesso = 99% *problema!*
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + minutos + microssegundos

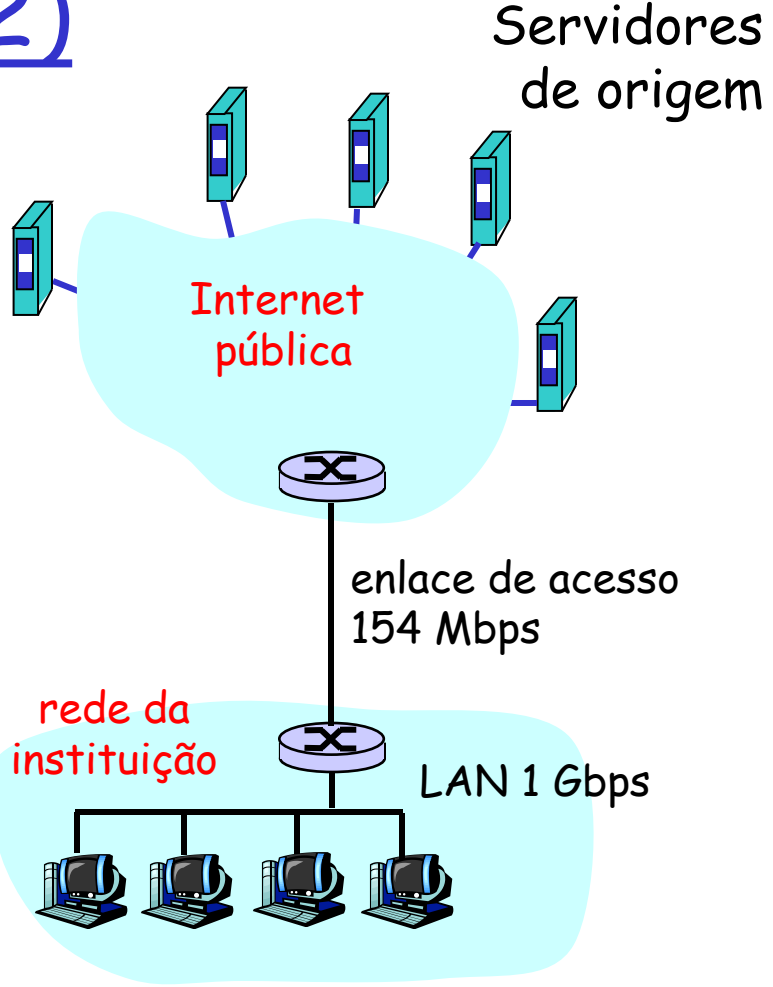
Exemplo de cache (2)

Solução em potencial

- Aumento da largura de banda do canal de acesso para, por exemplo, 154 Mbps

Consequências

- Utilização da LAN = 0,15%
- Utilização do canal de acesso = 9,9%
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + msegs + microssegundos
- Frequentemente este é uma ampliação cara



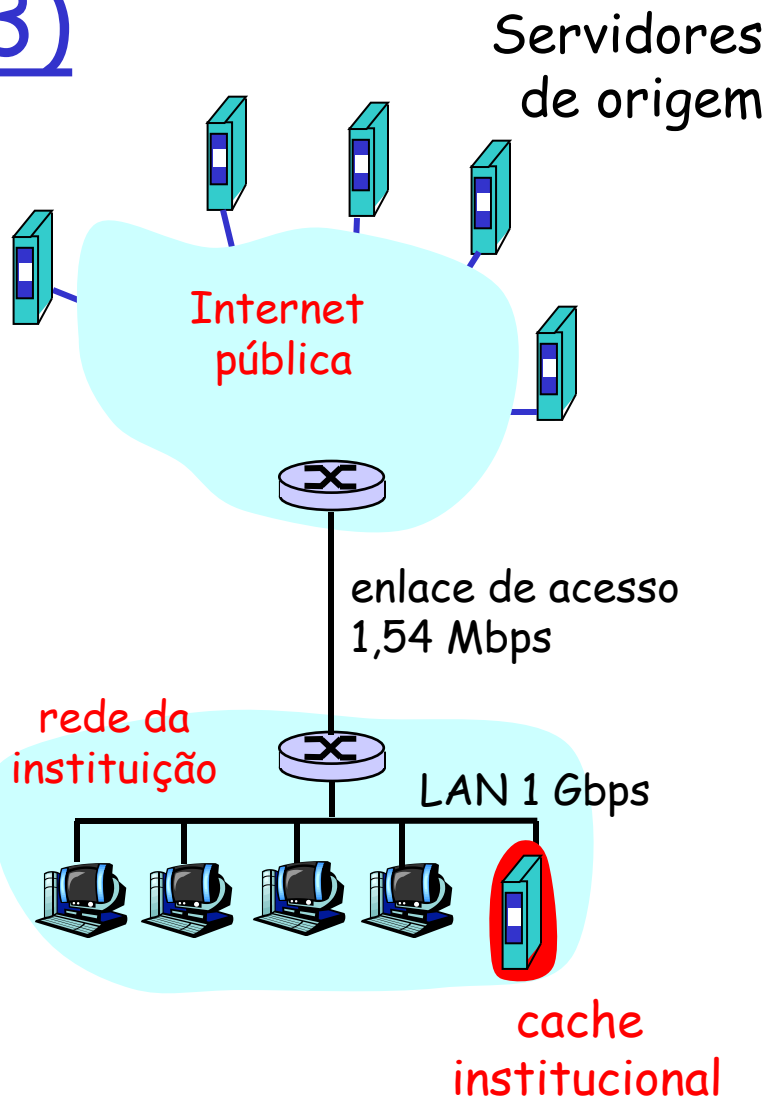
Exemplo de cache (3)

Instale uma cache

- Assuma que a taxa de acerto seja de 0,4

Consequências

- 40% dos pedidos serão atendidos quase que imediatamente
- 60% dos pedidos serão servidos pelos servidores de origem
- Utilização do canal de acesso é reduzido para 60%, resultando em atrasos desprezíveis (ex. 10 mseg)
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 0,6*2 seg + 0,6*0,01 segs + msegs < 1,3 segs



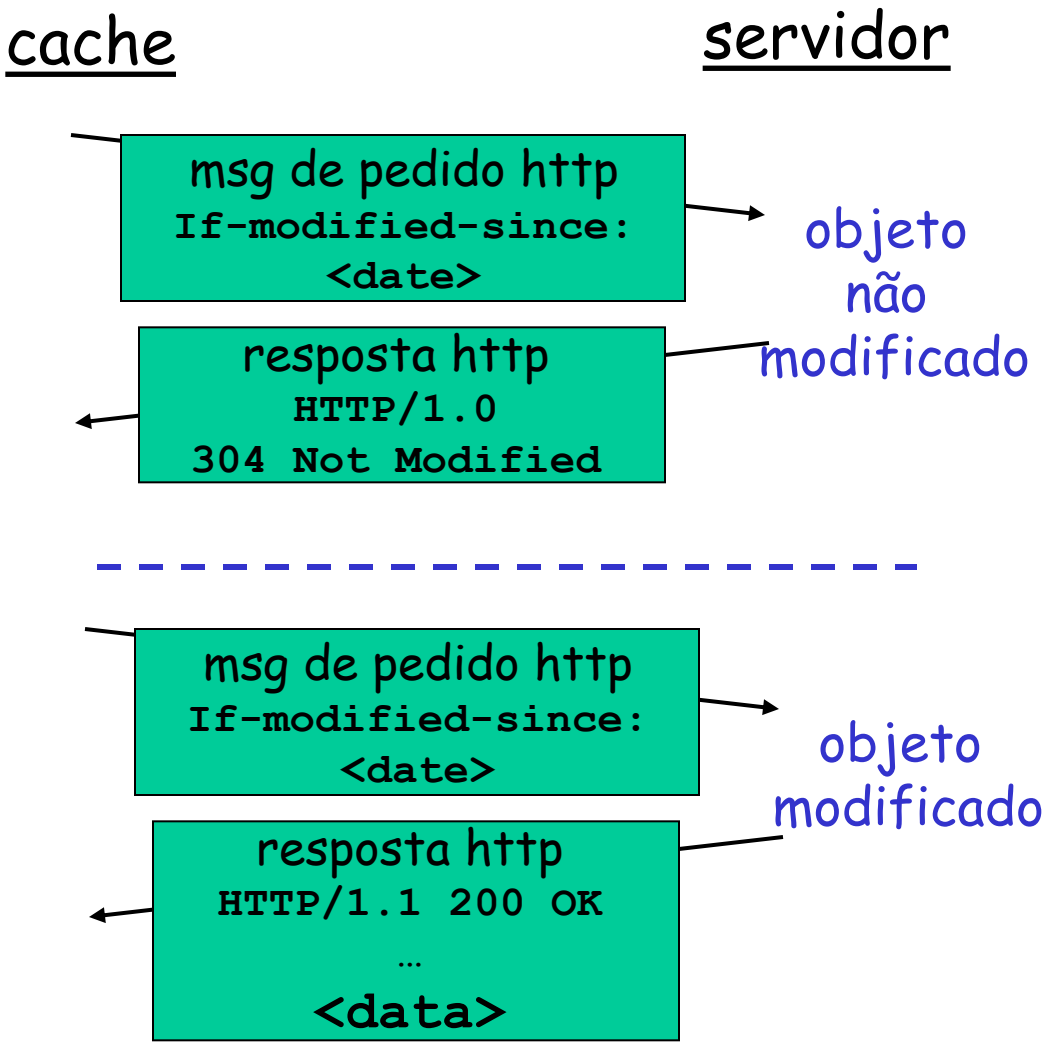
GET condicional

- Meta: não enviar objeto se cliente já tem (no cache) versão atual
  - Sem atraso para transmissão do objeto
  - Diminui a utilização do enlace
- cache: especifica data da cópia no cache no pedido HTTP

```
If-modified-since:
  <date>
```

- servidor: resposta não contém objeto se cópia no cache for atual:

```
HTTP/1.0 304 Not
  Modified
```



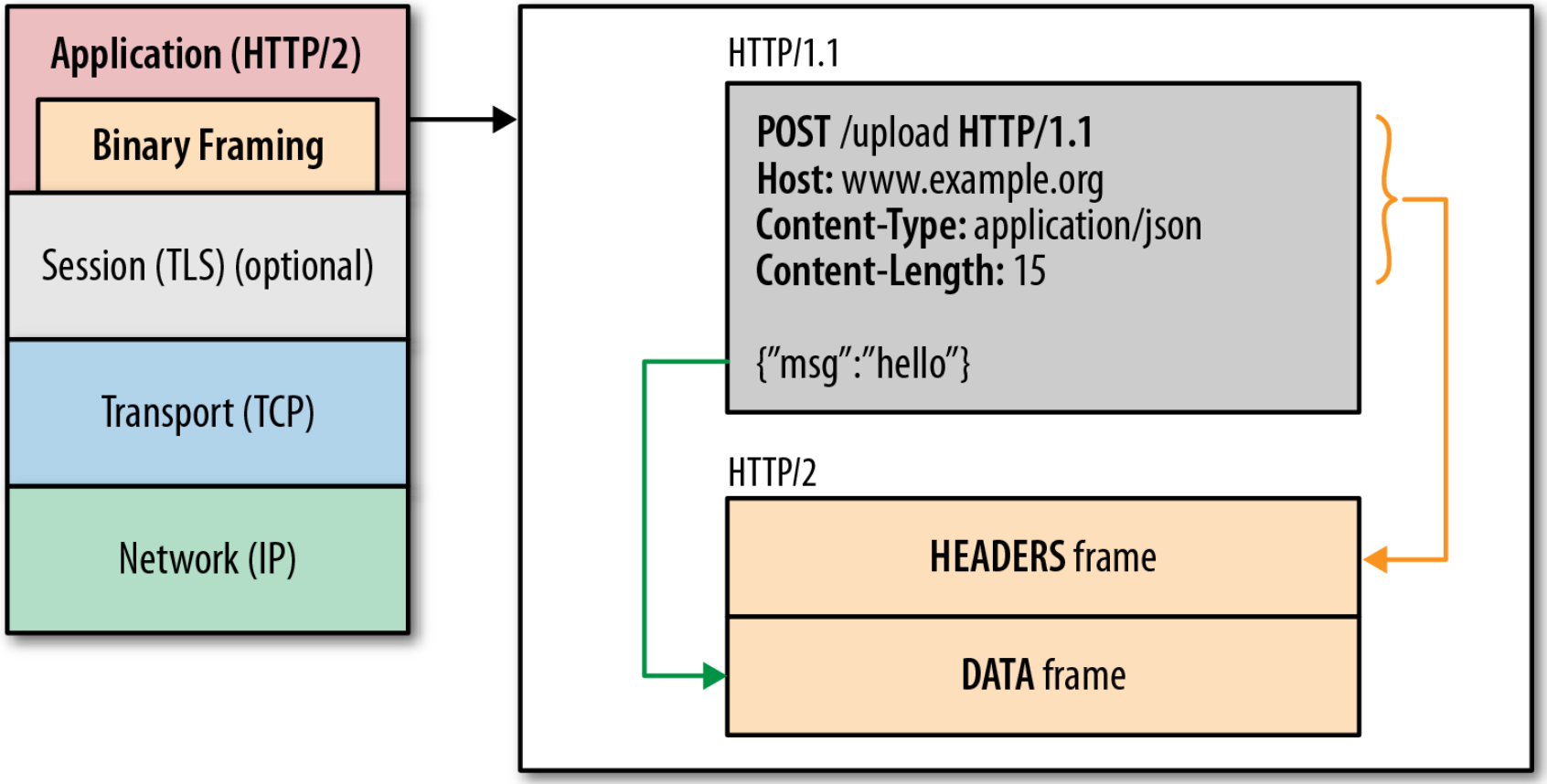
HTTP/2

- Aprovado pela IESG (*Internet Engineering Steering Group)* em Fevereiro de 2015
  - https://tools.ietf.org/html/draft-ietf-httpbis-http2-17
- Objetivos:
  - Mecanismos de negociação para permitir a clientes e servidores escolher o HTTP 1.1, 2, ou outros protocolos
  - Manutenção de compatibilidade de alto nível como HTTP 1.1
  - Diminuir a latência para melhorar a velocidade de carga das páginas através de:
    - Compressão de dados dos cabeçalhos HTTP
    - Tecnologias de envio (*push*) pelos servidores
    - Consertar o problema de bloqueio do cabeça da fila (HOL) do HTTP 1.1
    - Carga de elementos da página em paralelo através de uma única conexão TCP
  - Dar suporte aos casos de uso comuns atuais do HTTP

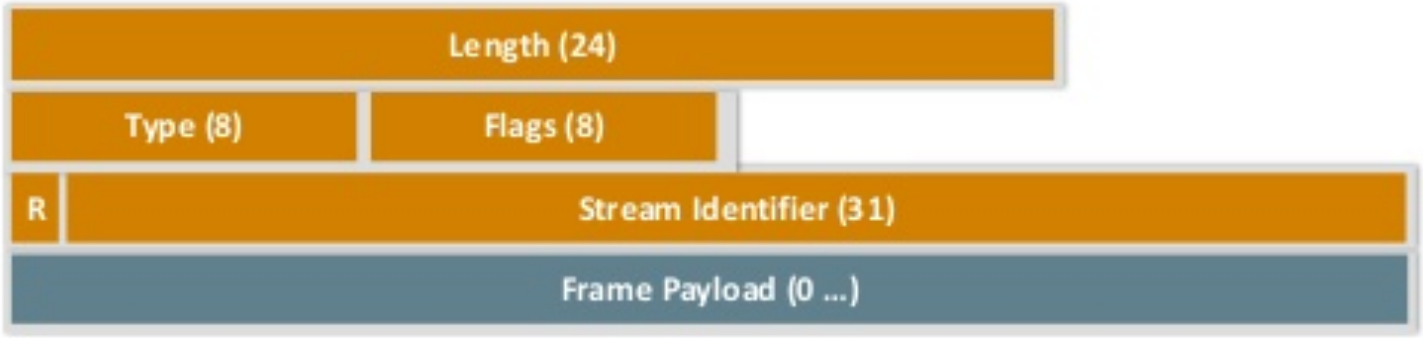
HTTP/2: Diferenças do HTTP 1.1

- Mantém a maior parte da sintaxe de alto nível do HTTP 1.1 tais como: métodos, códigos de status, campos de cabeçalhos e URIs
  - O que é modificado é como os dados são estruturados e transportados entre o cliente e o servidor de forma binária e não textual.
- HTTP/2 permite ao servidor enviar (*push*) conteúdo, i.e., enviar mais dados que os solicitados pelo cliente.
- Multiplexa os pedidos e as respostas para evitar o problema de bloqueio pelo cabeça da fila do HTTP 1.1.
- Realiza ainda um controle de fluxo e priorização dos pedidos.

HTTP/2: Transporte Binário

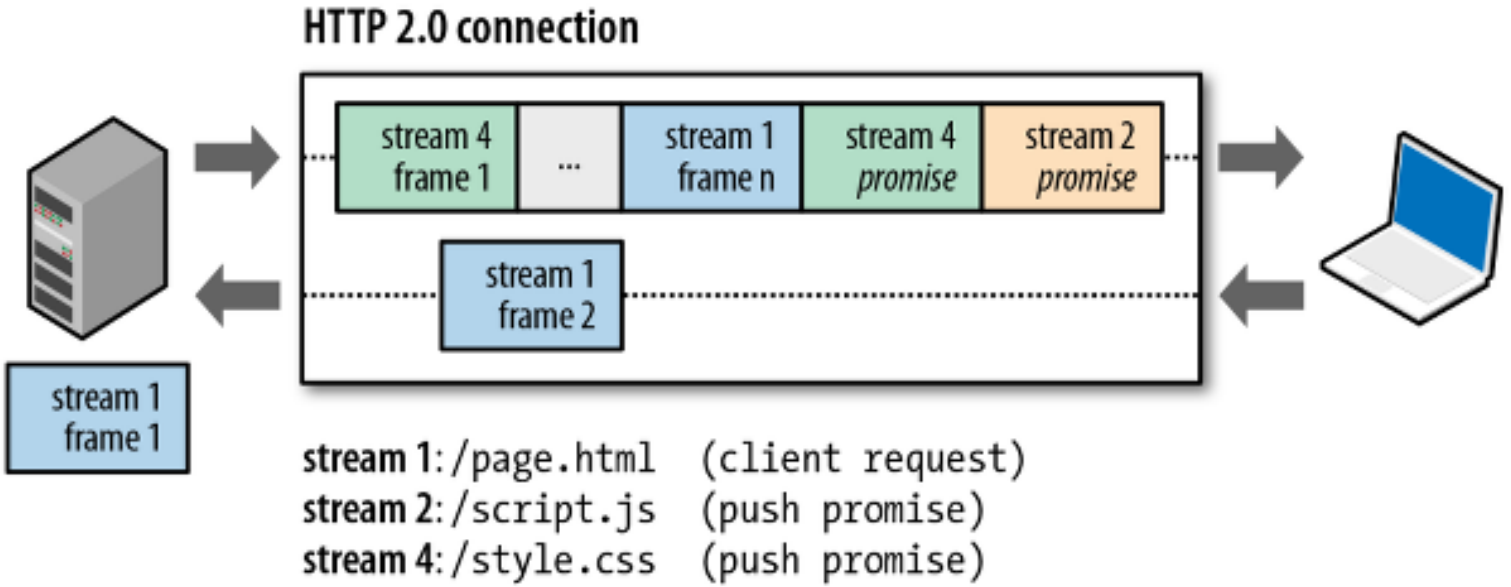


HTTP/2: Quadros



- Tipos:
  - HEADERS, DATA, PRIORITY, RST_STREAM, SETTINGS, PUSH_PROMISE, PING, GOAWAY, WINDOW_UPDATE, CONTINUATION

HTTP/2: Multiplexação



Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Correio Eletrônico na Internet
- 2.4 DNS: o serviço de diretório da Internet
- 2.5 Aplicações P2P
- 2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)
- 2.7 Programação de *sockets* com UDP e TCP

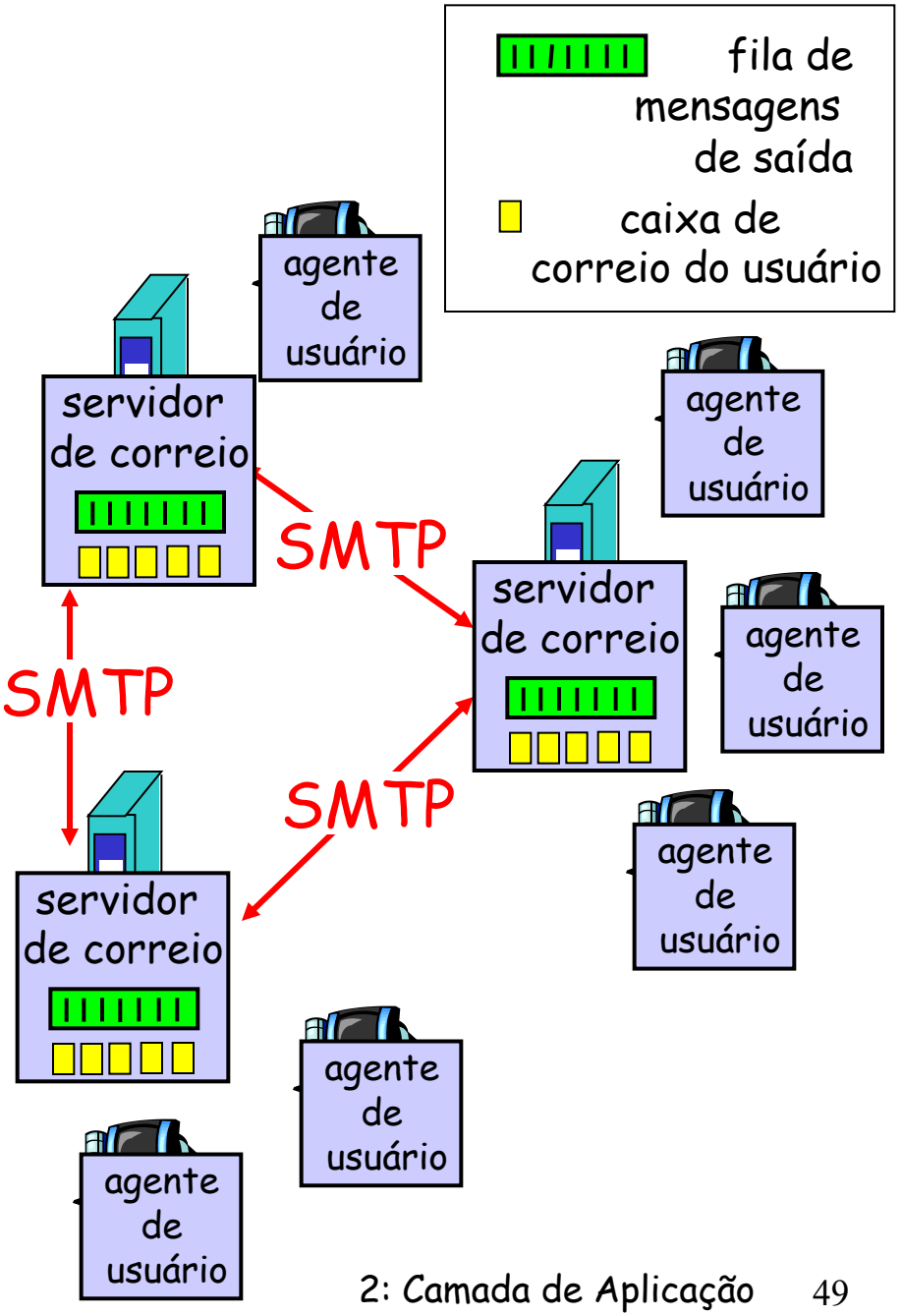
Correio Eletrônico

Três grandes componentes:

- agentes de usuário (UA)
- servidores de correio
- *Simple Mail Transfer Protocol*: SMTP

Agente de Usuário

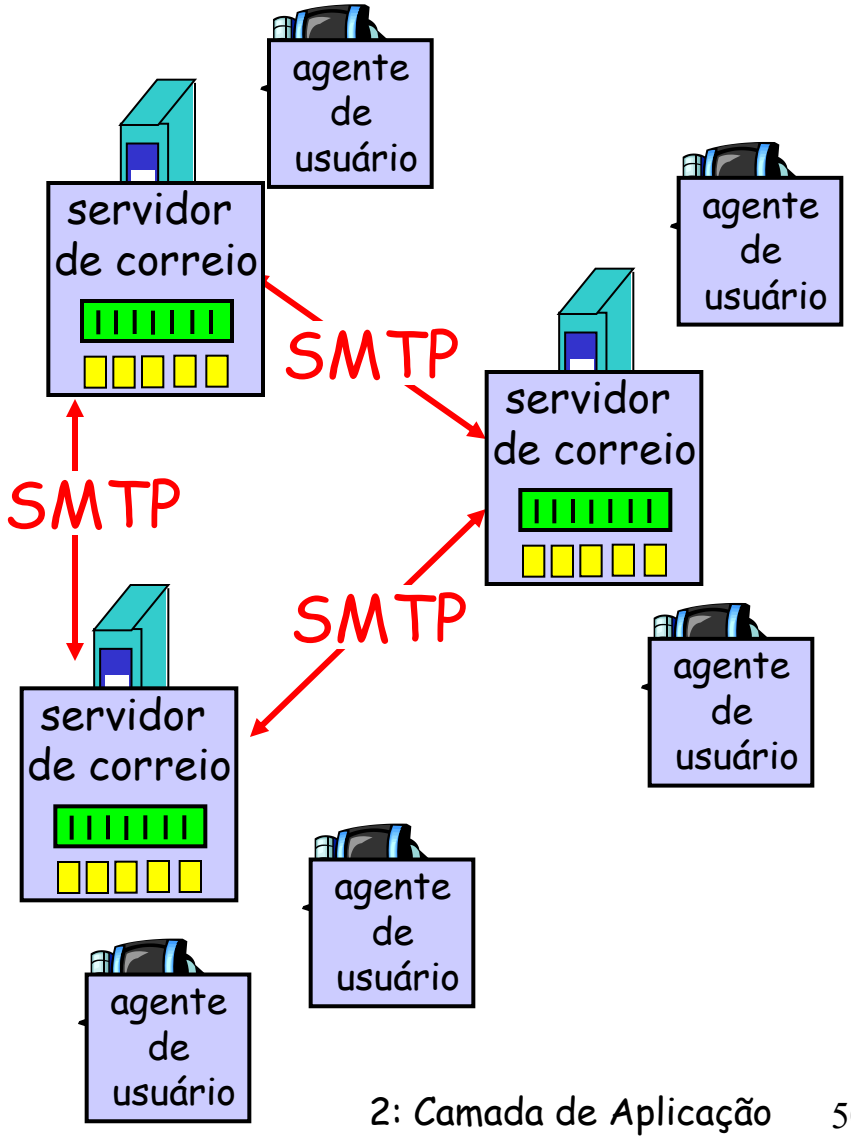
- a.k.a. "leitor de correio"
- compor, editar, ler mensagens de correio
- p.ex., Outlook, Thunderbird, cliente de mail do iPhone
- mensagens de saída e chegando são armazenadas no servidor



Correio Eletrônico: servidores de correio

Servidores de correio

- caixa de correio contém mensagens de chegada (ainda não lidas) p/ usuário
- fila de mensagens contém mensagens de saída (a serem enviadas)
- protocolo SMTP entre servidores de correio para transferir mensagens de correio
  - cliente: servidor de correio que envia
  - "servidor": servidor de correio que recebe



Correio Eletrônico: SMTP [RFC 2821]

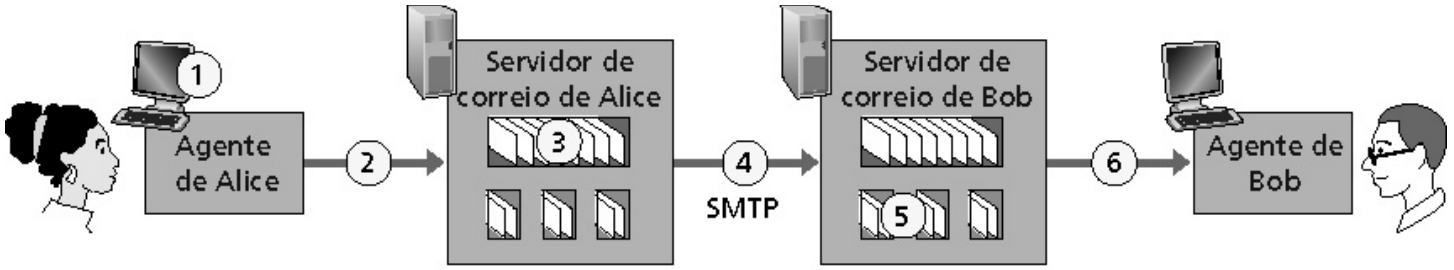
- usa TCP para a transferência confiável de msgs do correio do cliente ao servidor, porta 25
- transferência direta: servidor remetente ao servidor receptor
- três fases da transferência
  - *handshaking* (saudação)
  - transferência das mensagens
  - encerramento
- interação comando/resposta (como o HTTP e o FTP)
  - comandos: texto ASCII
  - resposta: código e frase de status
- mensagens precisam ser em ASCII de 7-bits

Gerência da Porta 25

http://antispam.br/

Cenário: Alice envia uma msg para Bob

1) Alice usa o UA para compor uma mensagem "para" `bob@someschool.edu`
2) O UA de Alice envia a mensagem para o seu servidor de correio; a mensagem é colocada na fila de mensagens
3) O lado cliente do SMTP abre uma conexão TCP com o servidor de correio de Bob
4) O cliente SMTP envia a mensagem de Alice através da conexão TCP
5) O servidor de correio de Bob coloca a mensagem na caixa de entrada de Bob
6) Bob chama o seu UA para ler a mensagem



Interação SMTP típica

```
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: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Experimente uma interação SMTP:

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

estes comandos permitem que você envie correio sem usar um cliente (leitor de correio)

SMTP: últimas palavras

- SMTP usa conexões persistentes
- SMTP requer que a mensagem (cabeçalho e corpo) sejam em ASCII de 7-bits
- servidor SMTP usa `CRLF.CRLF` para reconhecer o final da mensagem

Comparação com HTTP

- HTTP: *pull* (recupera)
- SMTP: *push* (envia)
- ambos têm interação comando/resposta, códigos de status em ASCII
- HTTP: cada objeto é encapsulado em sua própria mensagem de resposta
- SMTP: múltiplos objetos de mensagem enviados numa mensagem de múltiplas partes

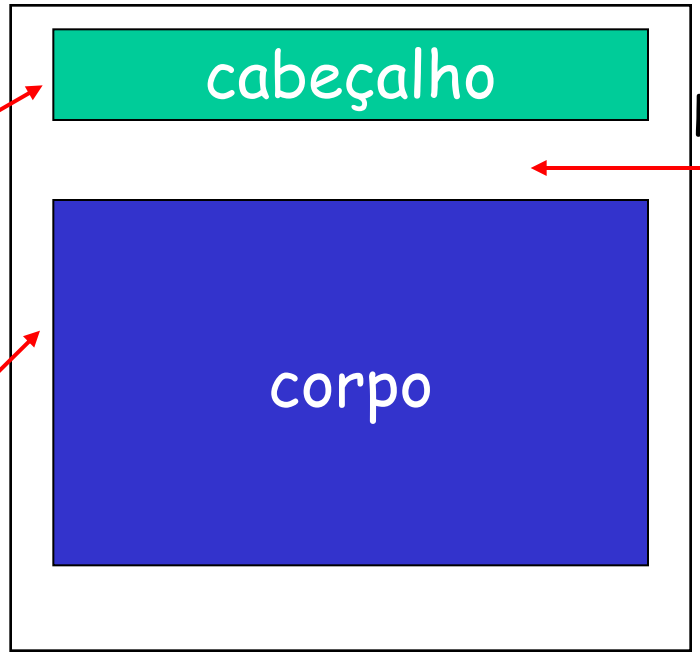
Formato de uma mensagem

SMTP: protocolo para trocar msgs de correio

RFC 822: padrão para formato de mensagem de texto:

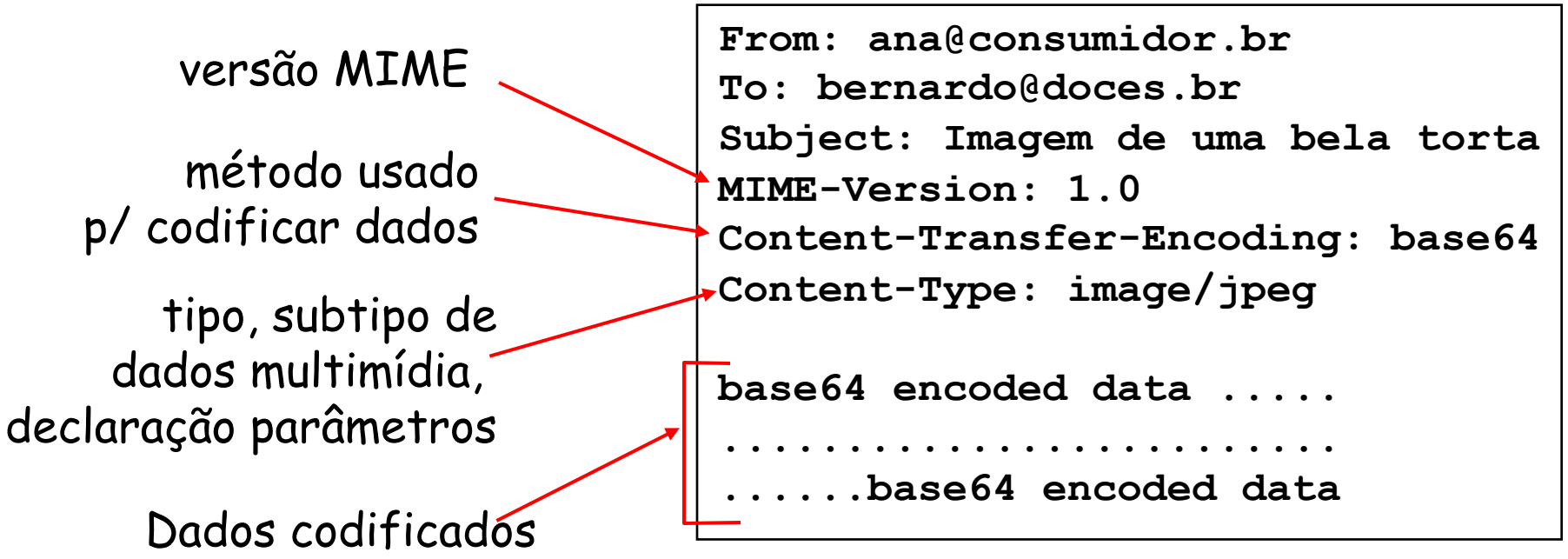
- linhas de cabeçalho, p.ex.,
  - To:
  - From:
  - Subject:

  diferentes dos comandos de smtp FROM, RCPT TO
- corpo
  - a "mensagem", somente de caracteres ASCII



Formato de uma mensagem: extensões para multimídia

- MIME: *multimedia mail extension*, RFC 2045, 2056
- linhas adicionais no cabeçalho da msg declaram tipo do conteúdo MIME



Tipos MIME

Content-Type: tipo/subtipo; parâmetros

Text

- subtipos exemplos: **plain, html**
- **charset="iso-8859-1", ascii**

Image

- subtipos exemplos : **jpeg, gif**

Video

- subtipos exemplos : **mpeg, quicktime**

Audio

- subtipos exemplos : **basic** (8-bit codificado mu-law), **32kadpcm** (codificação 32 kbps)

Application

- outros dados que precisam ser processados por um leitor para serem "visualizados"
- subtipos exemplos : **msword, octet-stream**

Tipo Multipart

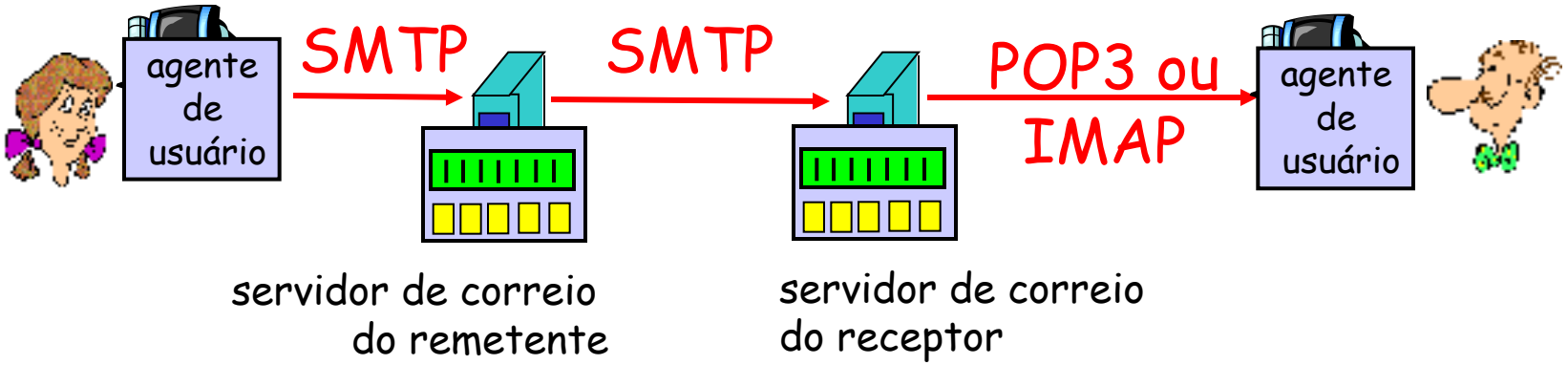
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```

Protocolos de acesso ao correio



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

Protocolo POP3

fase de autorização

- comandos do cliente:
  - **user**: declara nome
  - **pass**: senha
- servidor responde
  - **+OK**
  - **-ERR**

fase de transação, cliente:

- **list**: lista números das msgs
- **retr**: recupera msg por número
- **dele**: apaga msg
- **quit**

```
S: +OK POP3 server ready
C: user ana
S: +OK
C: pass faminta
S: +OK user successfully logged on
```

```
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 POP3 server signing off
```

POP3 (mais) e IMAP

Mais sobre o POP3

- O exemplo anterior usa o modo "*download* e delete".
- Bob não pode reler as mensagens se mudar de cliente
- "*Download*-e-mantenha": copia as mensagens em clientes diferentes
- POP3 não mantém estado entre conexões

IMAP

- Mantém todas as mensagens num único lugar: o servidor
- Permite ao usuário organizar as mensagens em pastas
- O IMAP mantém o estado do usuário entre sessões:
  - nomes das pastas e mapeamentos entre as IDs das mensagens e o nome da pasta

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Correio Eletrônico na Internet
- 2.4 DNS: o serviço de diretório da Internet
- 2.5 Aplicações P2P
- 2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)
- 2.7 Programação de *sockets* com UDP e TCP

DNS: *Domain Name System*

Pessoas: muitos identificadores:
- CPF, nome, no. da Identidade

hospedeiros, roteadores Internet :
- endereço IP (32 bit) - usado p/ endereçar datagramas
- "nome", ex., www.yahoo.com - usado por gente

P: como mapear entre nome e endereço IP?

Domain Name System:
- *base de dados distribuída* implementada na hierarquia de muitos *servidores de nomes*
- *protocolo de camada de aplicação* permite que hospedeiros, roteadores, servidores de nomes se comuniquem para *resolver* nomes (tradução endereço/nome)
  - nota: função imprescindível da Internet implementada como protocolo de camada de aplicação
  - complexidade na borda da rede

DNS (cont.)

Serviços DNS

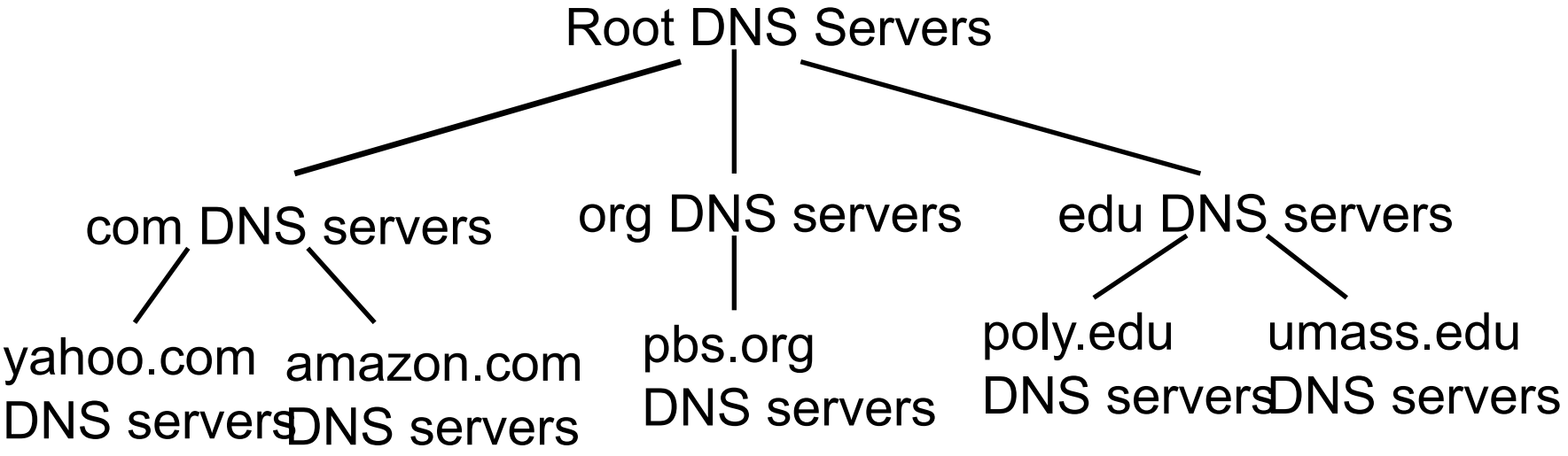
- Tradução de nome de hospedeiro para IP
- Apelidos para hospedeiros (*aliasing*)
  - Nomes canônicos e apelidos
- Apelidos para servidores de e-mail
- Distribuição de carga
  - Servidores Web replicados: conjunto de endereços IP para um mesmo nome

Por que não centralizar o DNS?

- ponto único de falha
- volume de tráfego
- base de dados centralizada e distante
- manutenção (da BD)

Não é escalável!

Base de Dados Hierárquica e Distribuída

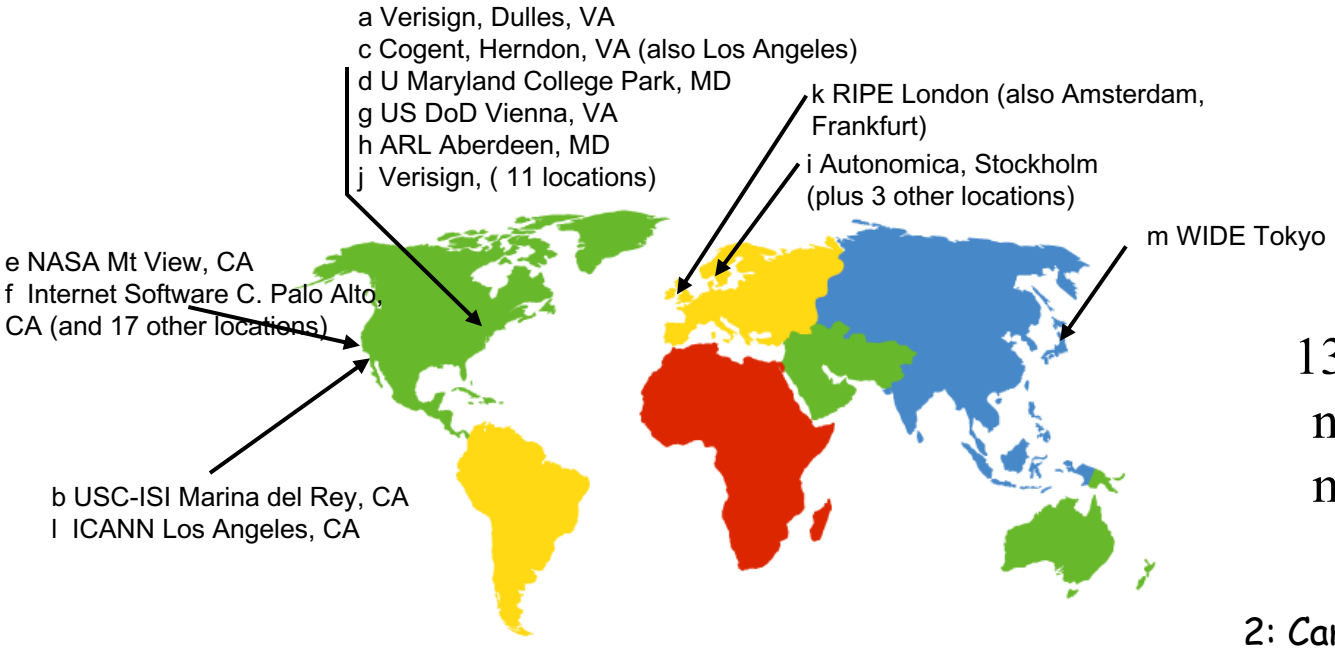


Cliente quer IP para www.amazon.com; 1ª aprox:

- Cliente consulta um servidor raiz para encontrar um servidor DNS .com
- Cliente consulta servidor DNS .com para obter o servidor DNS para o domínio amazon.com
- Cliente consulta servidor DNS do domínio amazon.com para obter endereço IP de www.amazon.com

DNS: Servidores raiz

- procurado por servidor local que não consegue resolver o nome
- servidor raiz:
  - procura servidor oficial se mapeamento desconhecido
  - obtém tradução
  - devolve mapeamento ao servidor local



13 servidores de nome raiz em todo o mundo

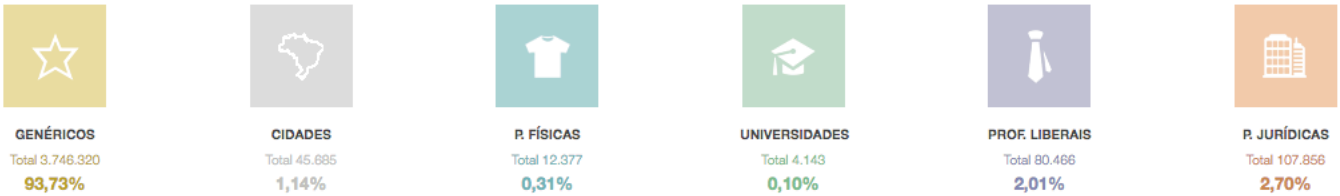
DNS: Servidores raiz

Hostname	IP Addresses	Manager
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	192.228.79.201, 2001:500:84::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

Servidores TLD e Oficiais

- Servidores de nomes de Domínio de Alto Nível (TLD):
  - servidores DNS responsáveis por domínios com, org, net, edu, etc, e todos os domínios de países como br, uk, fr, ca, jp.
  - Domínios genéricos: book, globo, rio
  - Lista completa em: https://www.iana.org/domains/root/db
  - NIC.br (Registro .br) para domínio .br (https://registro.br/)
- Servidores de nomes com autoridade:
  - servidores DNS das organizações, provendo mapeamentos oficiais entre nomes de hospedeiros e endereços IP para os servidores da organização (e.x., Web e correio).
  - Podem ser mantidos pelas organizações ou pelo provedor de acesso

Domínios registrados por categorias



21/09/2018

Genéricos

CATEGORIAS	QUANTIDADE	%
ART.BR	9.125	0,23
COM.BR	3.643.031	91,15
ECO.BR	9.485	0,24
EMP.BR	1.559	0,04
NET.BR	82.891	2,07
ONG.BR	229	0,01

» Ver evolução - total genéricos

Universidades

CATEGORIAS	QUANTIDADE	%
BR	1.207	0,03
EDU.BR	2.936	0,07

Cidades

CATEGORIAS	QUANTIDADE	%
9GUACU.BR	29	0,00
ABC.BR	1.331	0,03
AJU.BR	491	0,01
ANANI.BR	64	0,00
APARECIDA.BR	253	0,01
BARUERI.BR	118	0,00
BELEM.BR	554	0,01
BHZ.BR	1.830	0,05
BOAVISTA.BR	192	0,00
BSB.BR	2.704	0,07
CAMPINAGRANDE.BR	40	0,00
CAMPINAS.BR	2.060	0,05

VER TODOS

Pessoas Físicas

CATEGORIAS	QUANTIDADE	%
BLOG.BR	8.579	0,21
FLOG.BR	123	0,00
NOM.BR	1.309	0,03
VLOG.BR	1.081	0,03
WIKI.BR	1.285	0,03

» Ver evolução - total de pessoas físicas

21/09/2018

Profissionais liberais

CATEGORIAS	QUANTIDADE	%
ADM.BR	2.631	0,07
ADV.BR	34.591	0,87
ARQ.BR	5.703	0,14
ATO.BR	69	0,00
BIO.BR	480	0,01
BMD.BR	42	0,00
CIM.BR	610	0,02
CNG.BR	13	0,00
CNT.BR	2.882	0,07
ECN.BR	131	0,00

Pessoas Jurídicas

CATEGORIAS	QUANTIDADE	%
SEM RESTRIÇÃO		
AGR.BR	2.526	0,06
ESP.BR	1.161	0,03
ETC.BR	1.115	0,03
FAR.BR	482	0,01
IMB.BR	2.618	0,07
VER TODOS		
COM RESTRIÇÃO		
AM.BR	149	0,00
COOP.BR	1.050	0,03
FM.BR	368	0,01
G12.BR	591	0,01
GOV.BR	1.590	0,04
VER TODOS		
DNSSEC OBRIGATÓRIO		
B.BR	262	0,01
DEF.BR	25	0,00
JUS.BR	210	0,01
LEG.BR	59	0,00
MP.BR	34	0,00

Servidor DNS Local

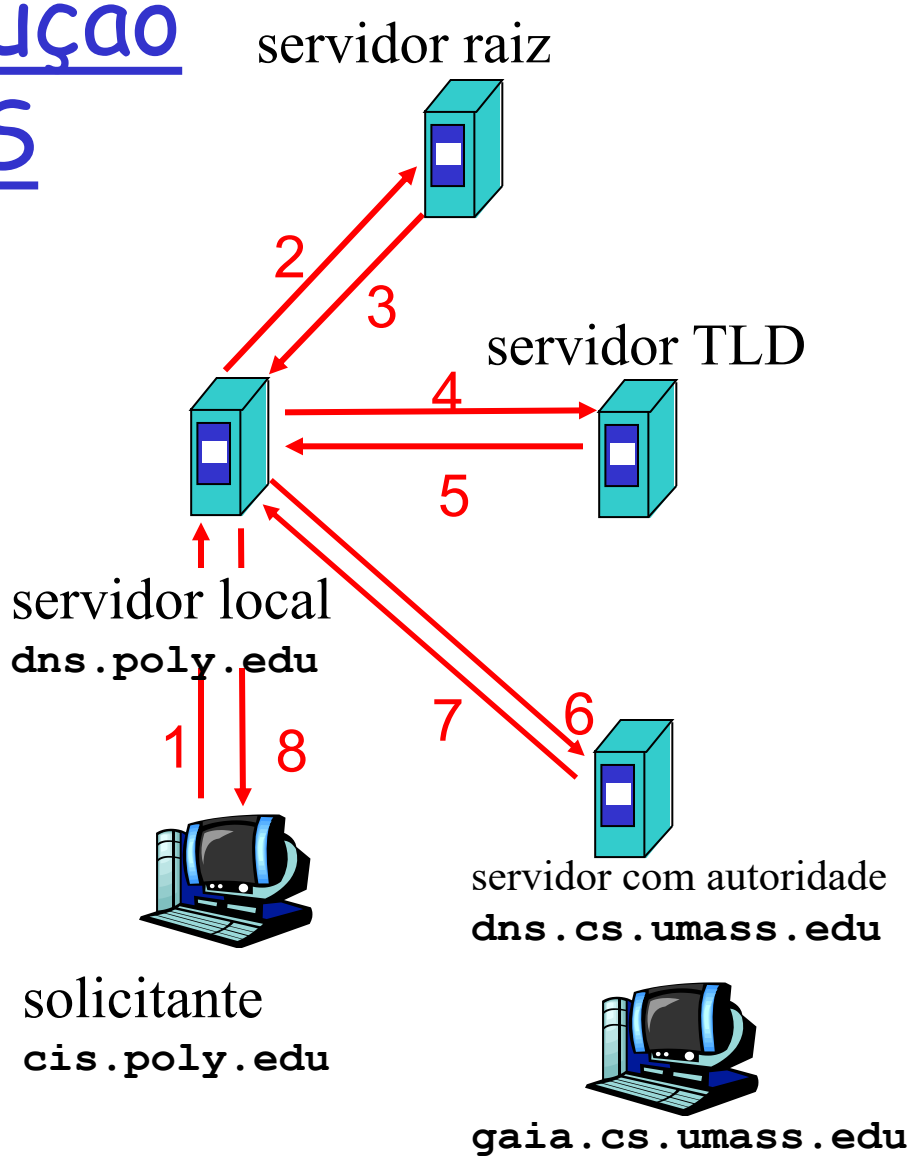
- Não pertence necessariamente à hierarquia
- Cada ISP (ISP residencial, companhia, universidade) possui um.
  - Também chamada do "servidor de nomes default"
- Quanto um hospedeiro faz uma consulta DNS, a mesma é enviada para o seu servidor DNS local
  - Possui uma cache local com pares de tradução nome/endereço recentes (mas podem estar desatualizados!)
  - Atua como um intermediário, enviando consultas para a hierarquia.

Exemplo de resolução de nome pelo DNS

- Hospedeiro em cis.poly.edu quer endereço IP para gaia.cs.umass.edu

consulta interativa:

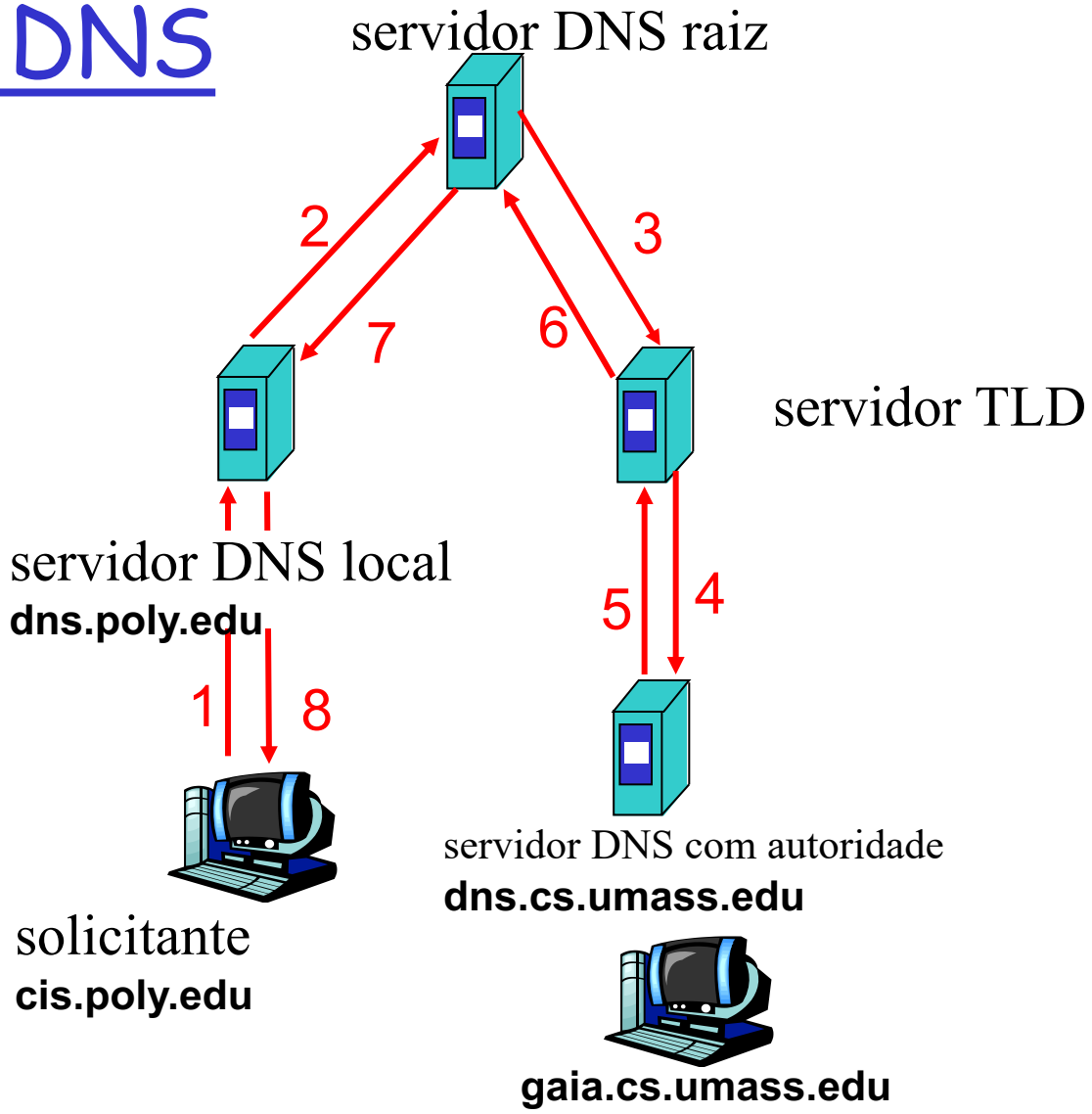
- servidor consultado responde com o nome de um servidor de contato
- "Não conheço este nome, mas pergunte para esse servidor"



Exemplo de resolução de nome pelo DNS

consulta recursiva:

- transfere a responsabilidade de resolução do nome para o servidor de nomes contatado
- carga pesada?



DNS: uso de cache, atualização de dados

- uma vez que um servidor qualquer aprende um mapeamento, ele o coloca numa *cache* local
  - entradas na cache são sujeitas a temporização (desaparecem) depois de um certo tempo (TTL)
- Entradas na cache podem estar desatualizadas (tradução nome/endereço do tipo melhor esforço!)
  - Se o endereço IP de um nome de host for alterado, pode não ser conhecido em toda a Internet até que todos os TTLs expirem
- mecanismos de atualização/notificação propostos na RFC 2136

Registros DNS

DNS: BD distribuído contendo *registros de recursos* (RR)

formato RR: **(nome, valor, tipo, ttl)**

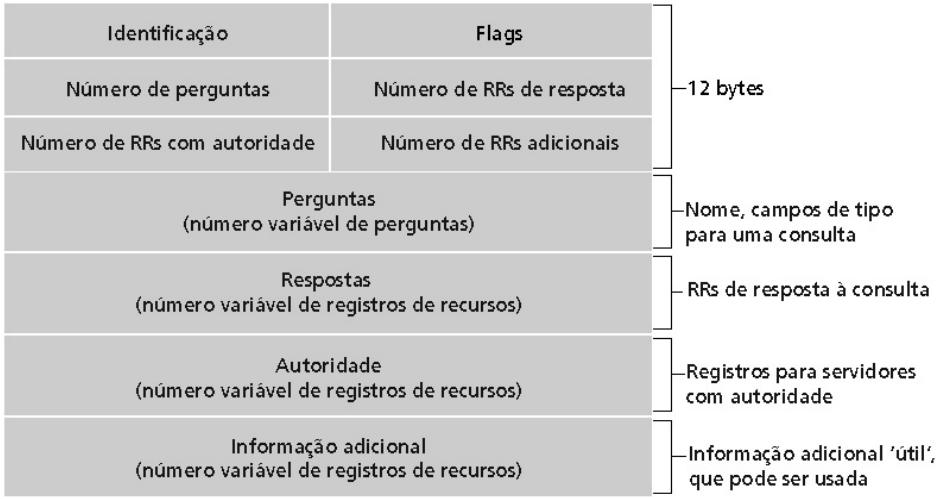
- Tipo=A
  - **nome** é nome de hospedeiro
  - **valor** é o seu endereço IPv4
  - Tipo=**AAAA** para IPv6
- Tipo=NS
  - **nome** é domínio (p.ex. foo.com.br)
  - **valor** é endereço IP de servidor oficial de nomes para este domínio
- Tipo=CNAME
  - **nome** é nome alternativo (alias) para algum nome "canônico" (verdadeiro)
  - www.ibm.com é na verdade servereast.backup2.ibm.com
  - **valor** é o nome canônico
- Tipo=MX
  - **valor** é nome do servidor de correio associado ao **nome**

DNS: protocolo e mensagens

protocolo DNS: mensagens de *pedido* e *resposta*, ambas com o mesmo *formato de mensagem*

cabeçalho de msg

- identificação: ID de 16 bit para pedido, resposta ao pedido usa mesmo ID
- flags:
  - pedido ou resposta
  - recursão desejada
  - recursão permitida
  - resposta é oficial



DNS: protocolo e mensagens

Inserindo registros no DNS

- Exemplo: acabou de criar a empresa "Network Utopia"
- Registra o nome netutopia.com.br em uma entidade registradora (e.x., Registro.br)
  - Tem de prover para a registradora os nomes e endereços IP dos servidores DNS oficiais (primário e secundário)
  - Registradora insere dois RRs no servidor TLD .br:

```
(netutopia.com.br, dns1.netutopia.com.br, NS)
(dns1.netutopia.com.br, 212.212.212.1, A)
```

- Põe no servidor oficial um registro do tipo A para www.netutopia.com.br e um registro do tipo MX para netutopia.com.br

Ataques ao DNS

Ataques DDoS

- Bombardeia os servidores raiz com tráfego
  - Até o momento não tiveram sucesso
  - Filtragem do tráfego
  - Servidores DNS locais cacheiam os IPs dos servidores TLD, permitindo que os servidores raízes não sejam consultados
- Bombardeio aos servidores TLD
  - Potencialmente mais perigoso

Ataques de redirecionamento

- Pessoa no meio
  - Intercepta as consultas
- Envenenamento do DNS
  - Envia respostas falsas para o servidor DNS que as coloca em cache

Exploração do DNS para DDoS

- Envia consultas com endereço origem falsificado: IP alvo
- Requer amplificação

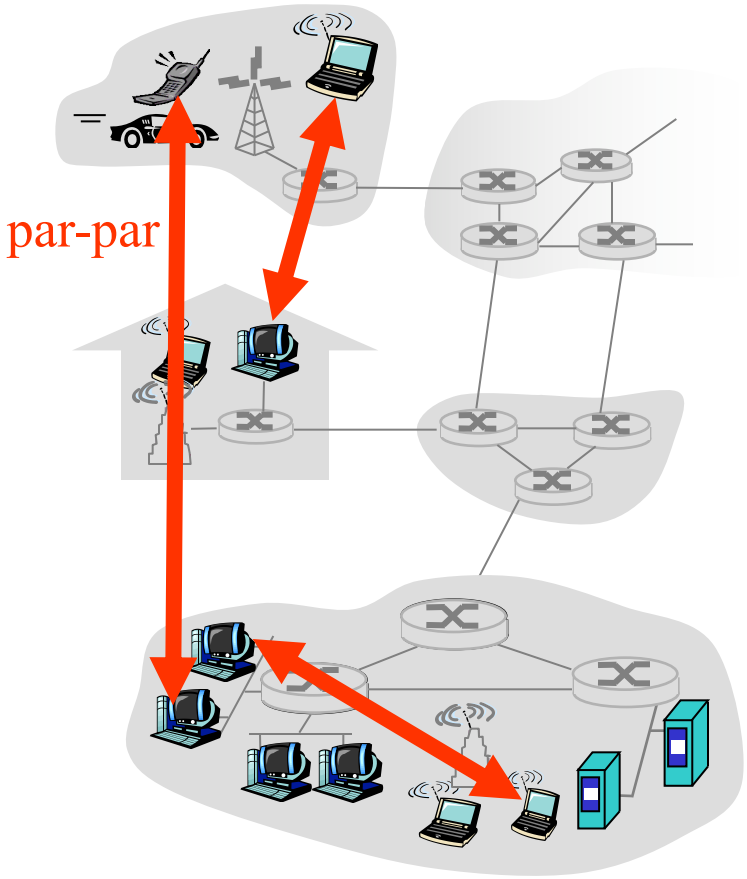
Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Correio Eletrônico na Internet
- 2.4 DNS: o serviço de diretório da Internet
- 2.5 Aplicações P2P
- 2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)
- 2.7 Programação de *sockets* com UDP e TCP

Arquitetura P2P pura

- *sem* servidor sempre ligado
- sistemas finais arbitrários se comunicam diretamente
- pares estão conectados de forma intermitente e mudam seus endereços IP

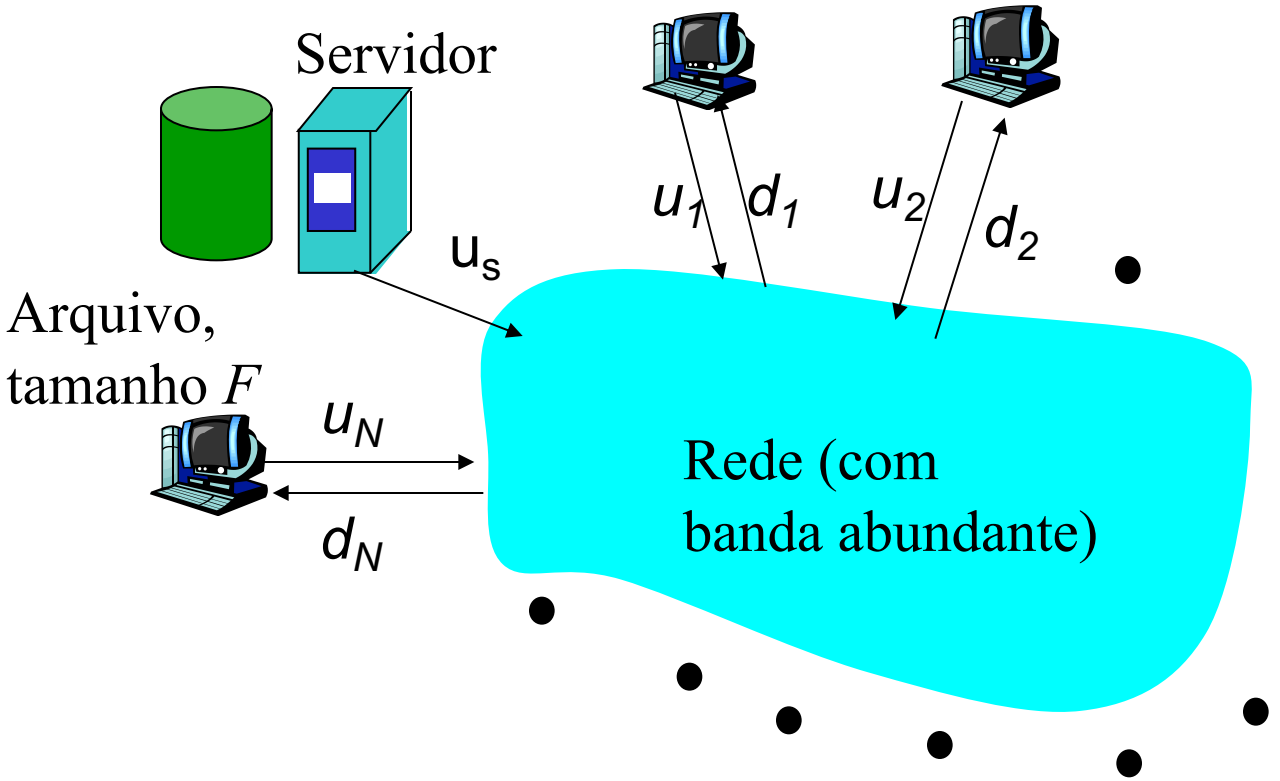
- Exemplos:
  - Distribuição de arquivos (BitTorrent)
  - *Streaming* (KanKan)
  - VoIP (Skype)



Distribuição de Arquivo: C/S x P2P

Pergunta: Quanto tempo leva para distribuir um arquivo de um servidor para *N* pares?

- Capacidade de *upload/download* de um par é um recurso limitado



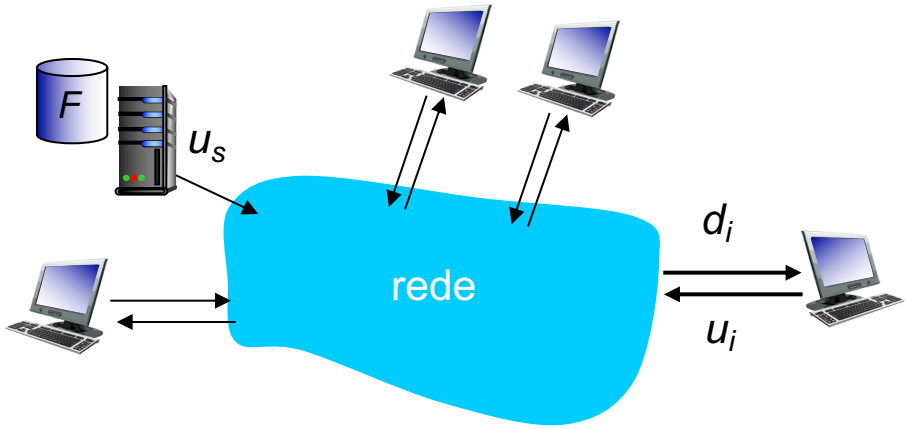
u_s: banda de upload do servidor

u_i: banda de upload do par *i*

d_i: banda de download do par *i*

Tempo de distribuição do arquivo: C/S

- **transmissão do servidor:** deve enviar sequencialmente N cópias do arquivo:
  - Tempo para enviar uma cópia = F/u_s
  - Tempo para enviar N cópias = NF/u_s
- **cliente:** cada cliente deve fazer o *download* de uma cópia do arquivo
  - d_{min} = taxa mínima de *download*
  - Tempo de *download* para usuário com menor taxa: F/d_{min}



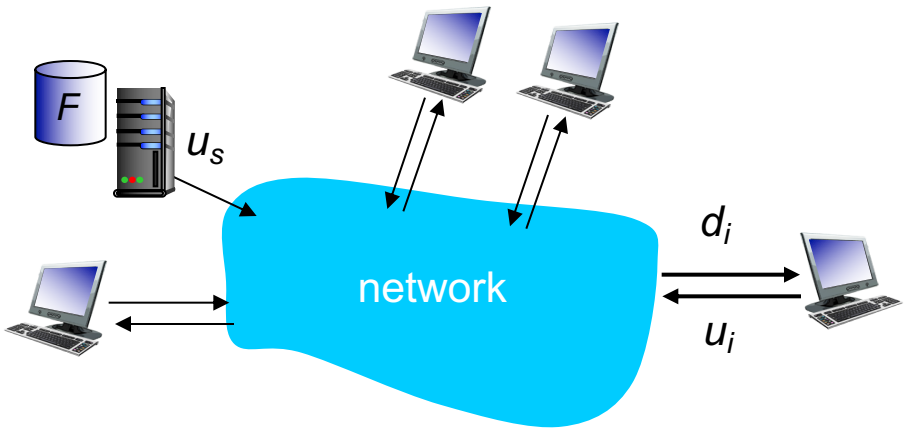
Tempo para distribuir F para N clientes usando abordagem cliente/servidor

$$D_{cs} \geq \max \{ NF/u_s, F/d_{min} \}$$

cresce linearmente com N

Tempo de distribuição do arquivo: P2P

- transmissão do servidor: deve enviar pelo menos uma cópia:
  - tempo para enviar uma cópia: F/u_s
- cliente: cada cliente deve baixar uma cópia do arquivo
  - Tempo de *download* para usuário com menor taxa: F/d_{min}
- clientes: no total devem baixar NF bits
  - Taxa máxima de *upload*: $u_s + \sum u_i$



tempo para distribuir F para N clientes usando abordagem P2P

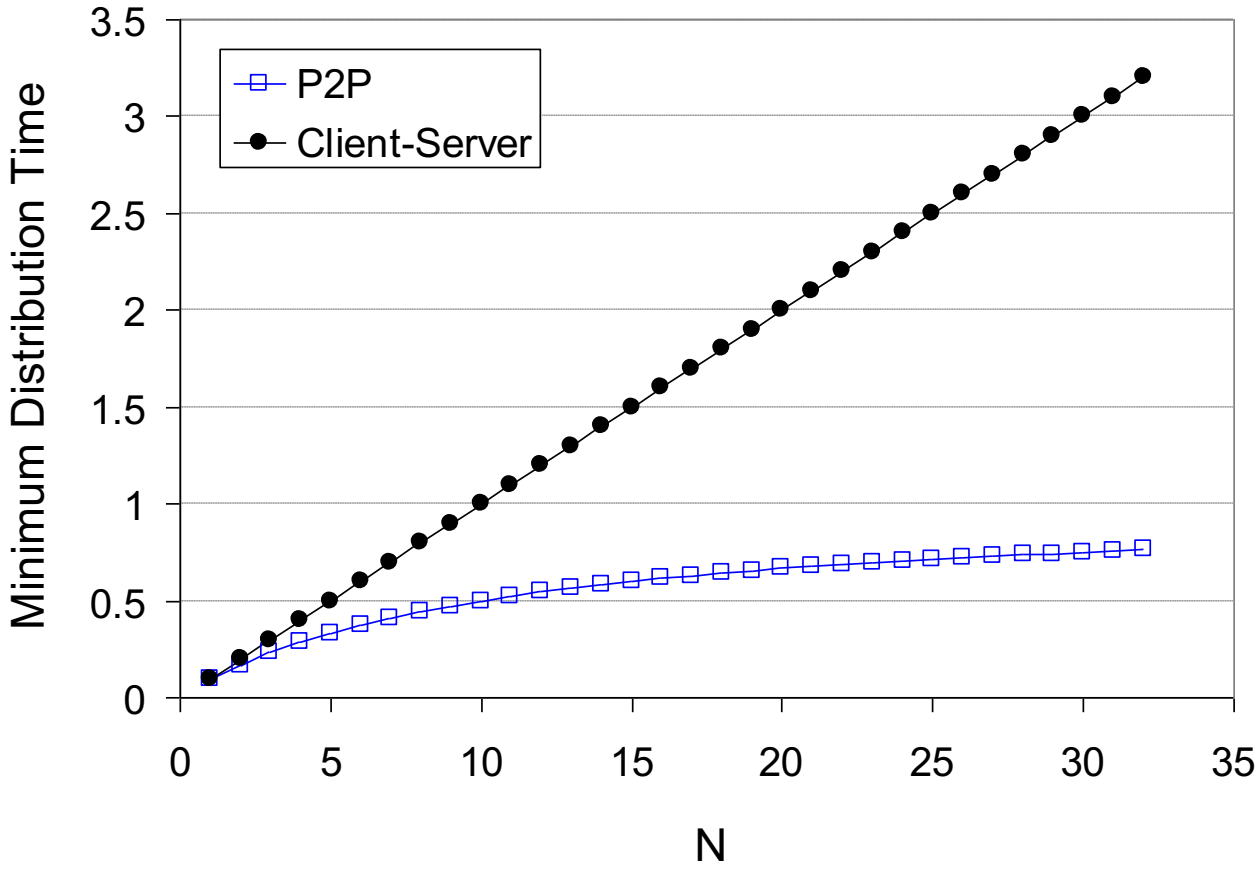
$$D_{P2P} > max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

cresce linearmente com *N* …

… assim como este, cada par traz capacidade de serviço

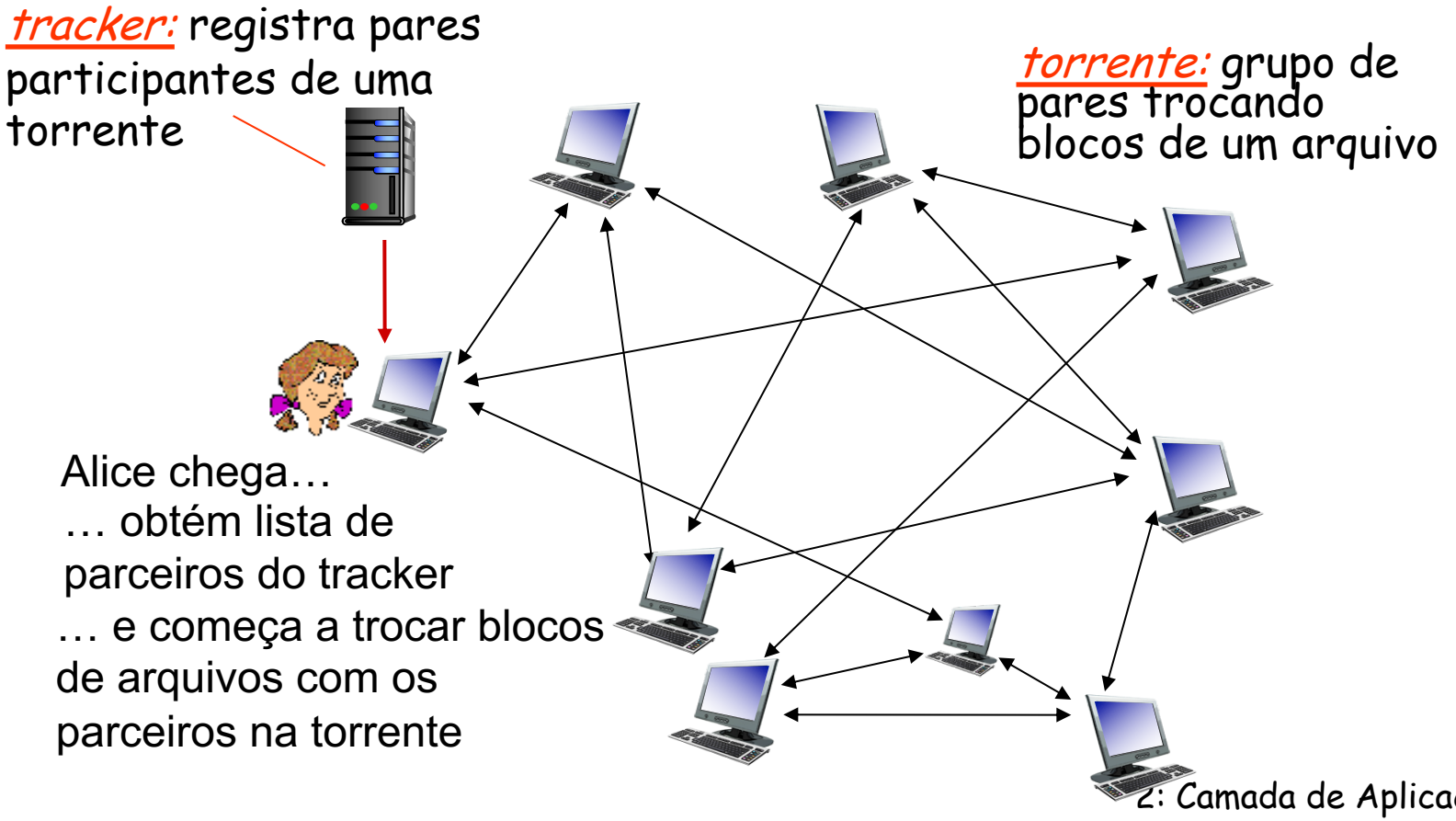
Cliente-servidor x P2P: Exemplo

Taxa de *upload* do cliente= u, F/u = 1 hora, u_s = 10u, $d_{min} \geq u_s$



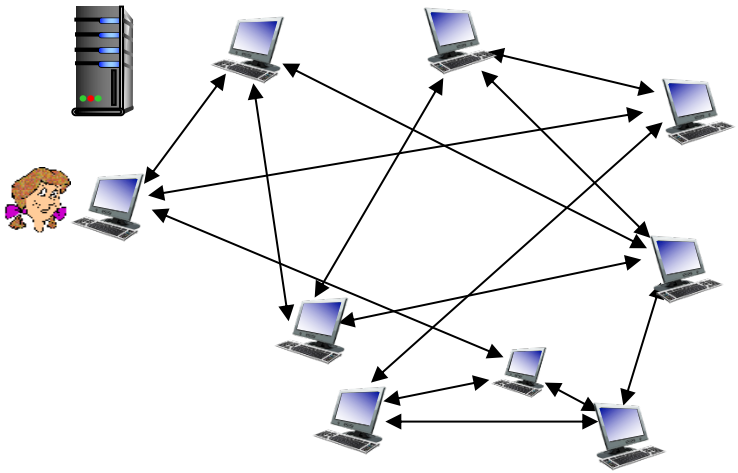
Distribuição de arquivo P2P: BitTorrent

- arquivos divididos em blocos de 256kb
- Pares numa torrente enviam/recebem blocos do arquivo



Distribuição de arquivo P2P: BitTorrent

- par que se une à torrente:
  - não tem nenhum bloco, mas irá acumulá-los com o tempo
  - registra com o *tracker* para obter lista dos pares, conecta a um subconjunto de pares ("vizinhos")



- enquanto faz o download, par carrega blocos para outros pares
- par pode mudar os parceiros com os quais troca os blocos
- pares podem entrar e sair
- quando o par obtiver todo o arquivo, ele pode (egoisticamente) sair ou permanecer (altruisticamente) na torrente

BitTorrent: pedindo, enviando blocos de arquivos

obtendo blocos:

- num determinado instante, pares distintos possuem diferentes subconjuntos de blocos do arquivo
- periodicamente, um par (Alice) pede a cada vizinho a lista de blocos que eles possuem
- Alice envia pedidos para os pedaços que ainda não tem
  - Primeiro os mais raros

Enviando blocos: toma lá, dá cá!

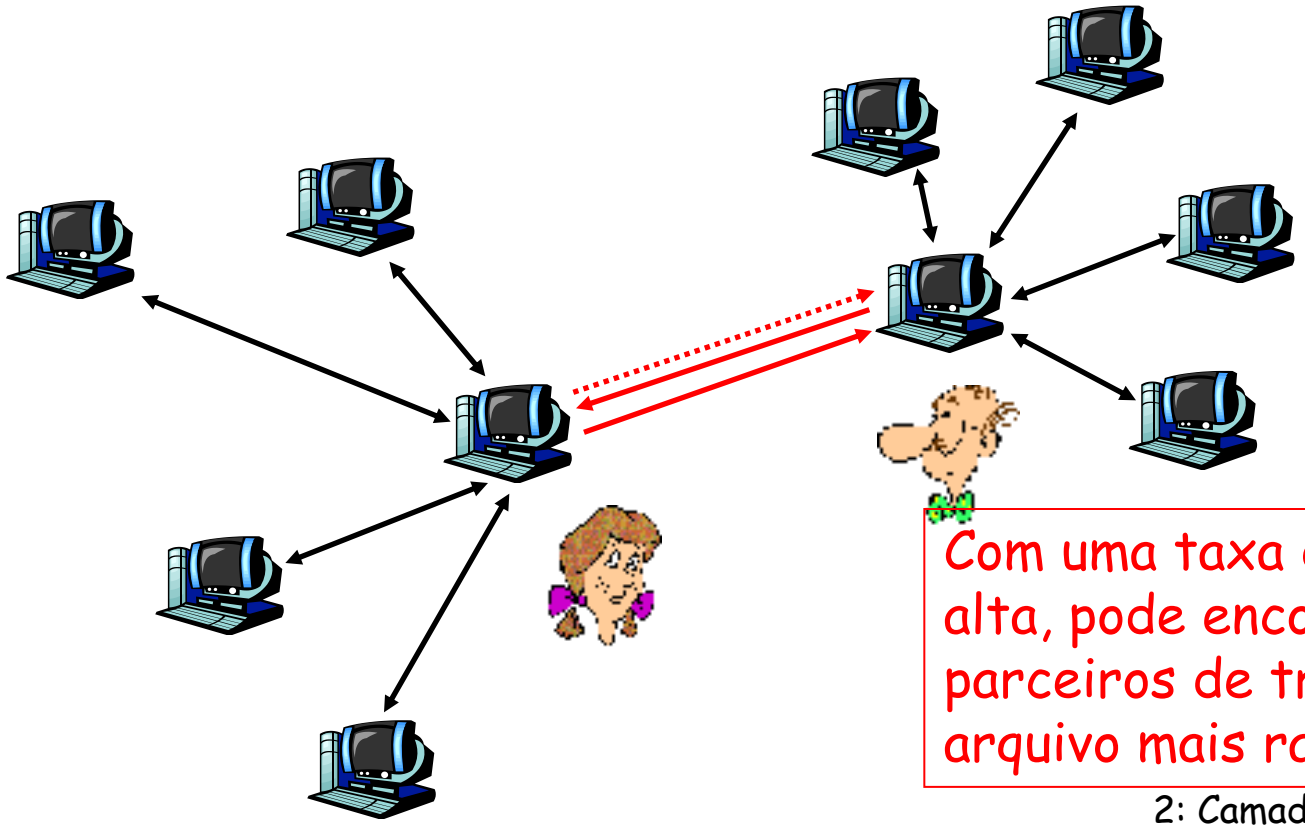
- Alice envia blocos para os quatro vizinhos que estejam lhe enviando blocos *na taxa mais elevada*
  - outros pares foram sufocados por Alice
  - Reavalia os 4 mais a cada 10 segs
- a cada 30 segs: seleciona aleatoriamente outro par, começa a enviar blocos
  - "*optimistically unchoked*"
  - o par recém escolhido pode se unir aos 4 mais

BitTorrent: toma lá, dá cá!

(1) Alice "optimistically unchokes" Bob

(2) Alice se torna um dos quatro melhores provedores de Bob; Bob age da mesma forma

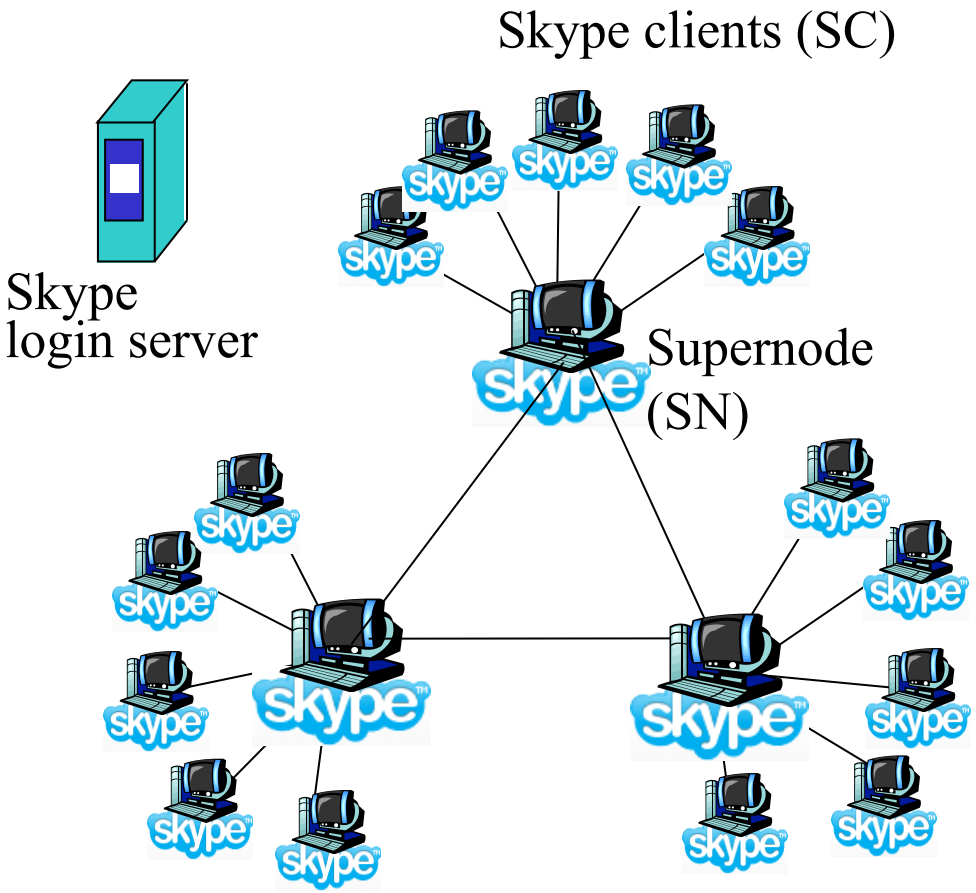
(3) Bob se torna um dos quatro melhores provedores de Alice



Com uma taxa de *upload* mais alta, pode encontrar melhores parceiros de troca e obter o arquivo mais rapidamente!

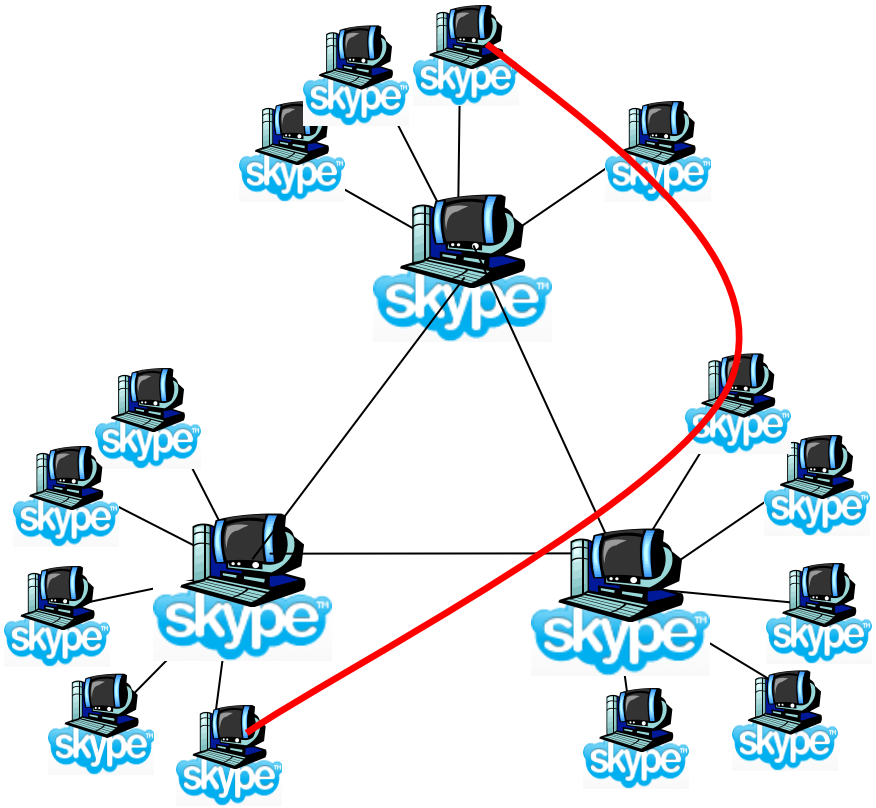
Estudo de caso P2P: Skype

- inerentemente P2P: comunicação entre pares de usuários.
- protocolo proprietário da camada de aplicação (inferido através de engenharia reversa)
- overlay hierárquico com SNs
- Índice mapeia nomes dos usuários a endereços IP; distribuído através dos SNs



Pares como intermediários (*relays*)

- Problema quando tanto Alice como Bob estão atrás de "NATs".
  - O NAT impede que um par externo inicie uma chamada com um par interno
- Solução:
  - Intermediário é escolhido, usando os SNs de Alice e de Bob.
  - Cada par inicia sessão com o intermediário
  - Pares podem se comunicar através de NATs através do intermediário



Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Correio Eletrônico na Internet
- 2.4 DNS: o serviço de diretório da Internet
- 2.5 Aplicações P2P
- 2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)
- 2.7 Programação de *sockets* com UDP e TCP

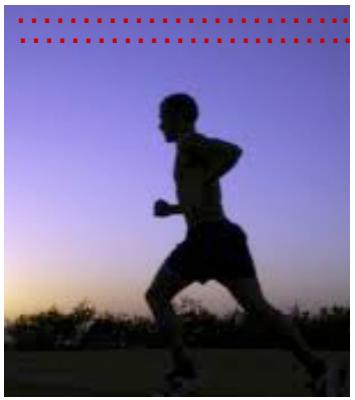
Streaming de vídeo e CDNs: contexto

- Tráfego de vídeo: maior consumidor de largura de banda da Internet
  - Netflix, YouTube: 37%, 16% do tráfego *downstream* residencial dos ISPs
  - ~1B usuários do YouTube, ~75M usuários do Netflix
- desafio: escala – como alcançar ~1B usuários?
  - um único mega-vídeo server não daria conta (por quê?)
- desafio: heterogeneidade
  - usuários diferentes têm diferentes características (ex.: cabeado x móvel; boa x ruim largura de banda)
- *solução:* infraestrutura distribuída de camada de aplicação

Multimídia: vídeo

- vídeo: sequência de imagens apresentadas a uma taxa constante
  - e.g., 24 imagens/seg
- Imagem digital : matriz de pixels
  - cada pixel representado por bits
- codificação: usa redundância *dentro* e *entre* imagens para diminuir # bits usados para codificar a imagem
  - espacial (dentro da imagem)
  - temporal (de uma imagem para a próxima)

exemplo de codificação espacial: ao invés de enviar N valores com a mesma cor (roxo), envia apenas dois valores: valor da cor (roxo) e número (N) de valores repetidos (N)



quadro *i*

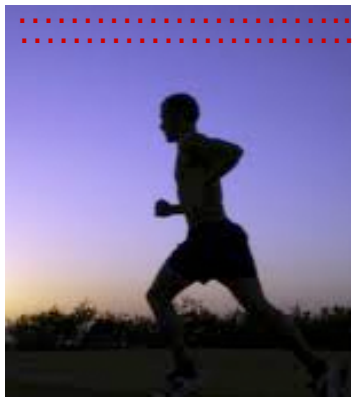
quadro *i+1*

exemplo de codificação temporal: ao invés de enviar o quadro completo i+1, envia apenas as diferenças do quadro i

Multimídia: vídeo

- CBR (*constant bit rate*): codificação de vídeo a taxa constante
- VBR (*variable bit rate*): taxa de codificação de video muda com a necessidade/redundância espacial ou temporal.
- exemplos:
  - MPEG 1 (CD-ROM) 1,5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (frequentemente usado na Internet, < 1 Mbps)

exemplo de codificação espacial: ao invés de enviar N valores com a mesma cor (roxo), envia apenas dois valores: valor da cor (roxo) e número (N) de valores repetidos (N)



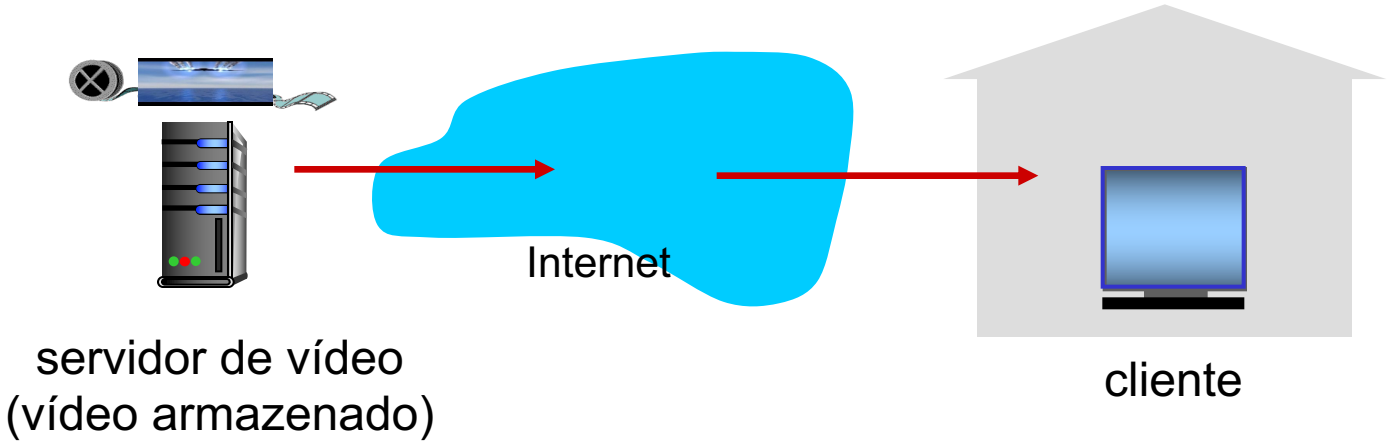
quadro *i*

quadro *i+1*

exemplo de codificação temporal: ao invés de enviar o quadro completo i+1, envia apenas as diferenças do quadro i

Streaming de vídeo armazenado:

cenário simples:



Streaming multimídia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *servidor:*
  - divide o arquivo de vídeo em diversos pedaços (*chunks*)
  - cada pedaço é armazenado codificado em diferentes taxas
  - *arquivo de manifesto:* provê URLs para os diferentes pedaços
- *cliente:*
  - mede periodicamente a banda entre servidor e cliente
  - consulta manifesto, solicita um pedaço por vez
    - escolhe a taxa máxima suportada pela largura de banda atual
    - pode escolher diferentes taxas de codificação em instantes diferentes (dependendo a banda disponível no momento)

Streaming multimídia: DASH

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- *"inteligência"* no cliente: o cliente determina
  - *quando* solicitar um pedido (de modo a não haver nem esvaziamento nem estouro do *buffer*)
  - *que taxa de codificação* solicitar (maior qualidade quando houver mais banda disponível)
  - *de onde* solicitar o pedaço (pode solicitar do servidor URL que estiver mais "próximo" do cliente ou que tenha a maior banda disponível)

Redes de Distribuição de Conteúdo (CDNs)

- *desafio:* como enviar conteúdo (selecionado de milhões de vídeos) para centenas de milhares de usuários *simultâneos*?

- *opção 1:* grande "mega-servidor" único
  - ponto único de falha
  - ponto de congestionamento de rede
  - caminho longo distante dos clientes
  - múltiplas cópias do vídeo enviadas pelo link de saída

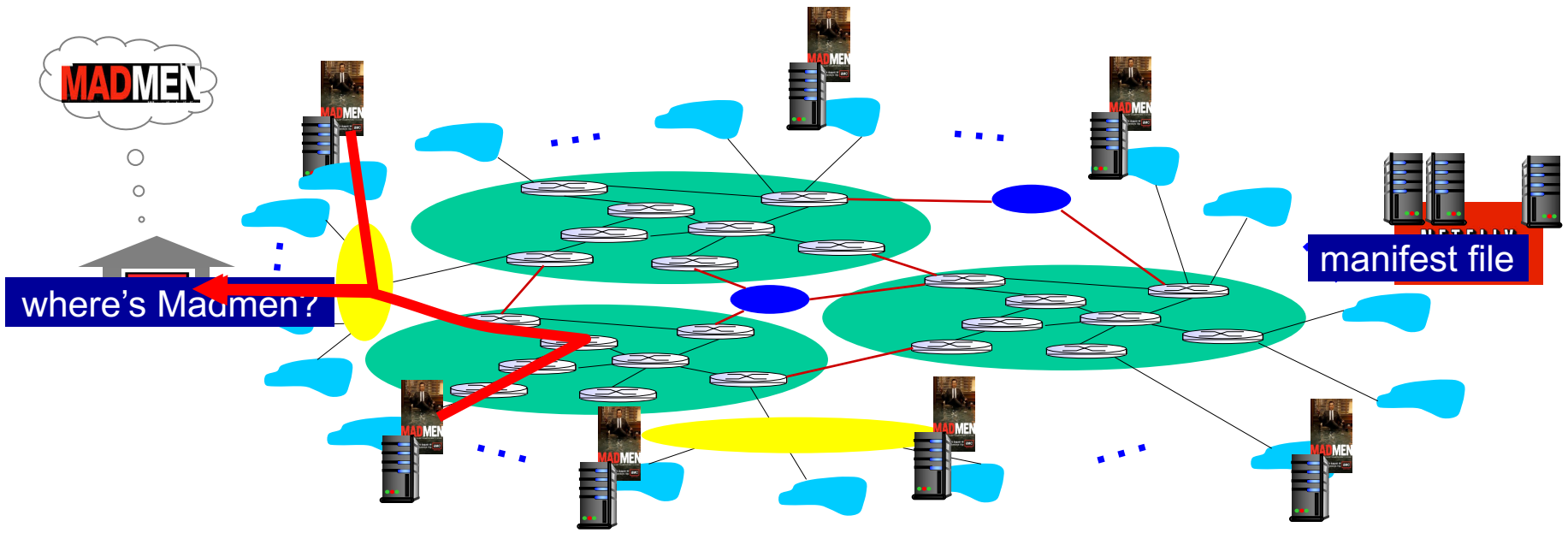
….simplesmente: esta solução *não escala*

Redes de Distribuição de Conteúdo (CDNs)

- *desafio:* como enviar conteúdo (selecionado de milhões de vídeos) para centenas de milhares de usuários *simultâneos*?
- *opção 2:* armazenar/disponibilizar múltiplas cópias do vídeo em sites distribuídos geograficamente *(CDN)*
  - *ir fundo:* colocar servidores CDN em muitas redes de acesso
    - próximo aos usuários
    - usado pela Akamai, 1700 localidades
  - *levar para casa:* menor número (10's) de grandes clusters em POPs próximos (mas não dentro) das redes de acesso
    - usado pela Limelight

Redes de Distribuição de Conteúdos (CDNs)

- CDN: armazena cópias do conteúdo em nós
  - e.g. Netflix armazena cópias de MadMen
- assinante solicita conteúdo da CDN
  - é direcionado para cópia mais próxima, recupera o conteúdo
  - pode escolher outra cópia, se o caminho estiver congestionado



Redes de Distribuição de Conteúdos (CDNs)

desafios: convivendo com uma Internet congestionada

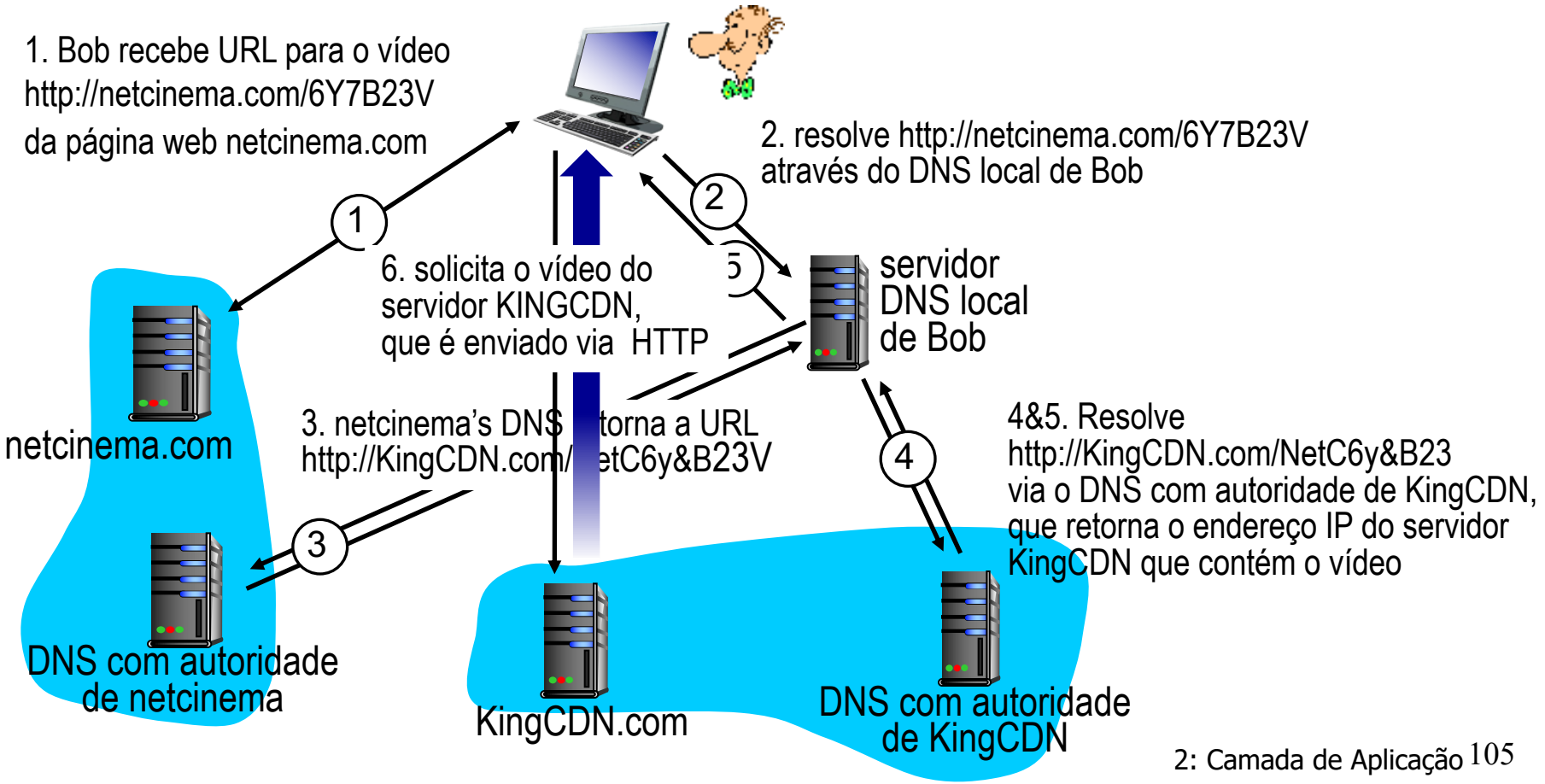
- de qual nó CDN se deve recuperar o conteúdo?
- comportamento do usuário na presença de congestionamento?
- que conteúdo colocar em cada nó CDN?

mais .. no capítulo 7

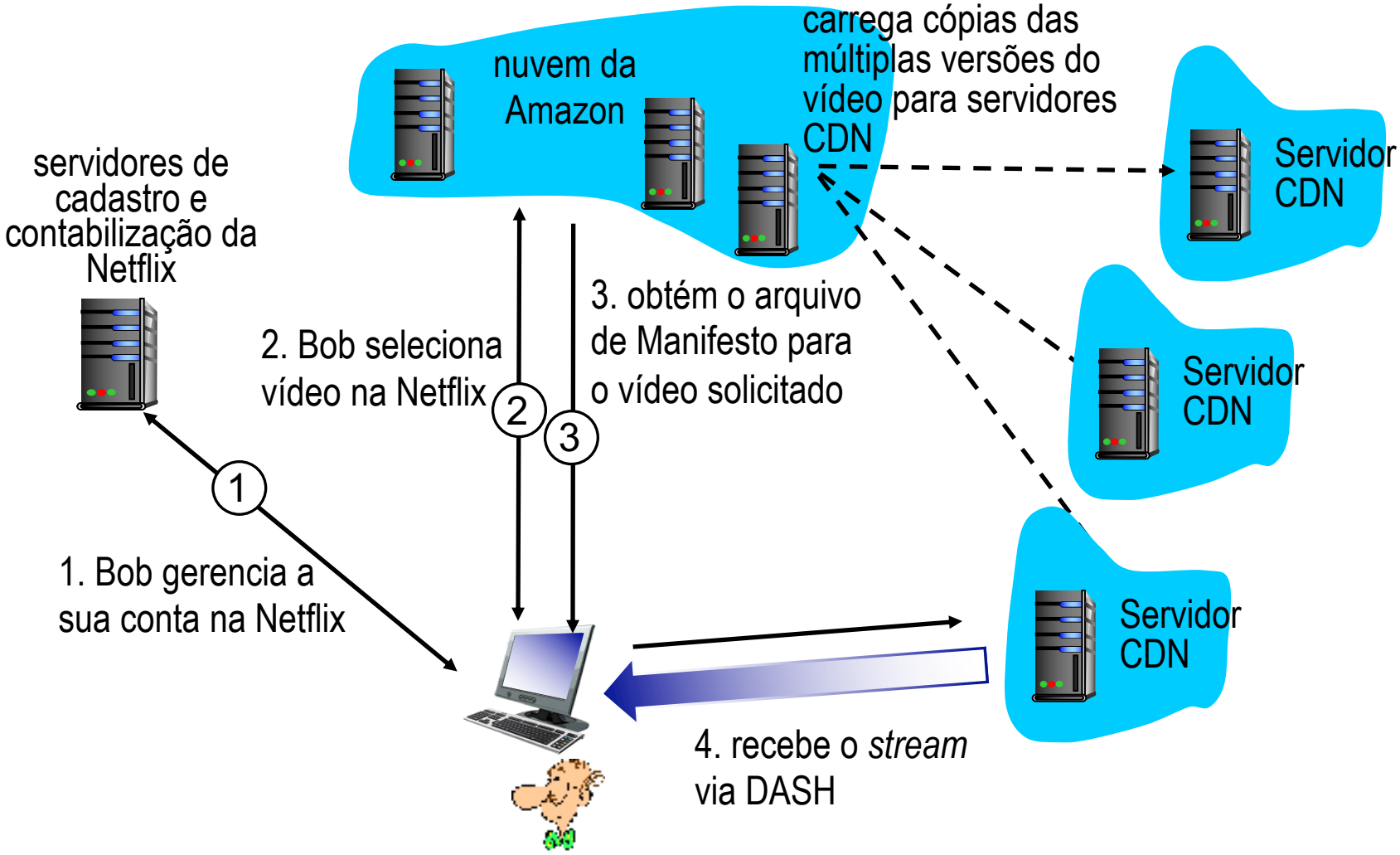
acesso a conteúdo CDN: detalhes

Bob (cliente) solicita o vídeo http://netcinema.com/6Y7B23V

- vídeo armazenado na CDN em http://KingCDN.com/NetC6y&B23V



Caso de estudo: Netflix



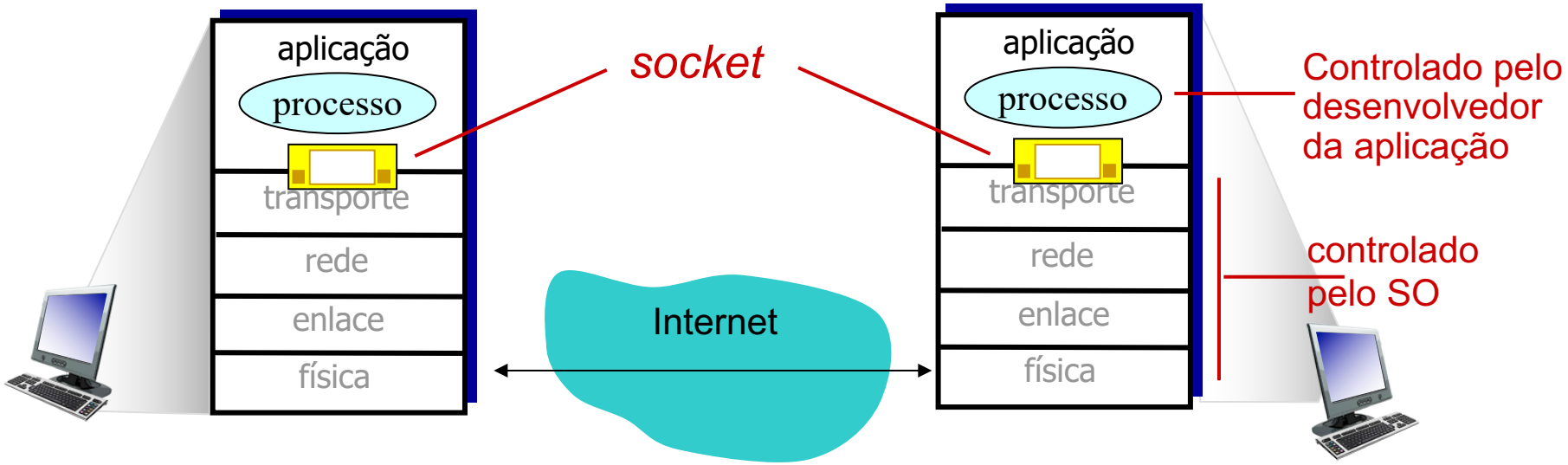
Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Correio Eletrônico na Internet
- 2.4 DNS: o serviço de diretório da Internet
- 2.5 Aplicações P2P
- 2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)
- 2.7 Programação de *sockets* com UDP e TCP

Programação com sockets

meta: aprender a construir aplicações cliente/servidor que se comunicam usando sockets

socket: porta entre o processo de aplicação e o protocolo de transporte fim-a-fim



Programação com sockets

Dois tipos de sockets para dois serviços de transporte:

- *UDP:* datagrama não confiável
- *TCP:* confiável, orientado a fluxos de bytes

Aplicação Exemplo:

1. o cliente lê uma linha de caracteres (dados) do seu teclado e envia os dados para o servidor
2. o servidor recebe os dados e converte os caracteres para maiúsculas
3. o servidor envia os dados modificados para o cliente
4. o cliente recebe os dados modificados e apresenta a linha na sua tela

Programação com sockets usando UDP

UDP: não tem "conexão" entre cliente e servidor

- não tem saudação ("handshaking") antes de enviar os dados
- remetente coloca explicitamente endereço IP e porta do destino
- servidor deve extrair endereço IP e número da porta do remetente do datagrama recebido

UDP: dados transmitidos podem ser recebidos fora de ordem, ou perdidos

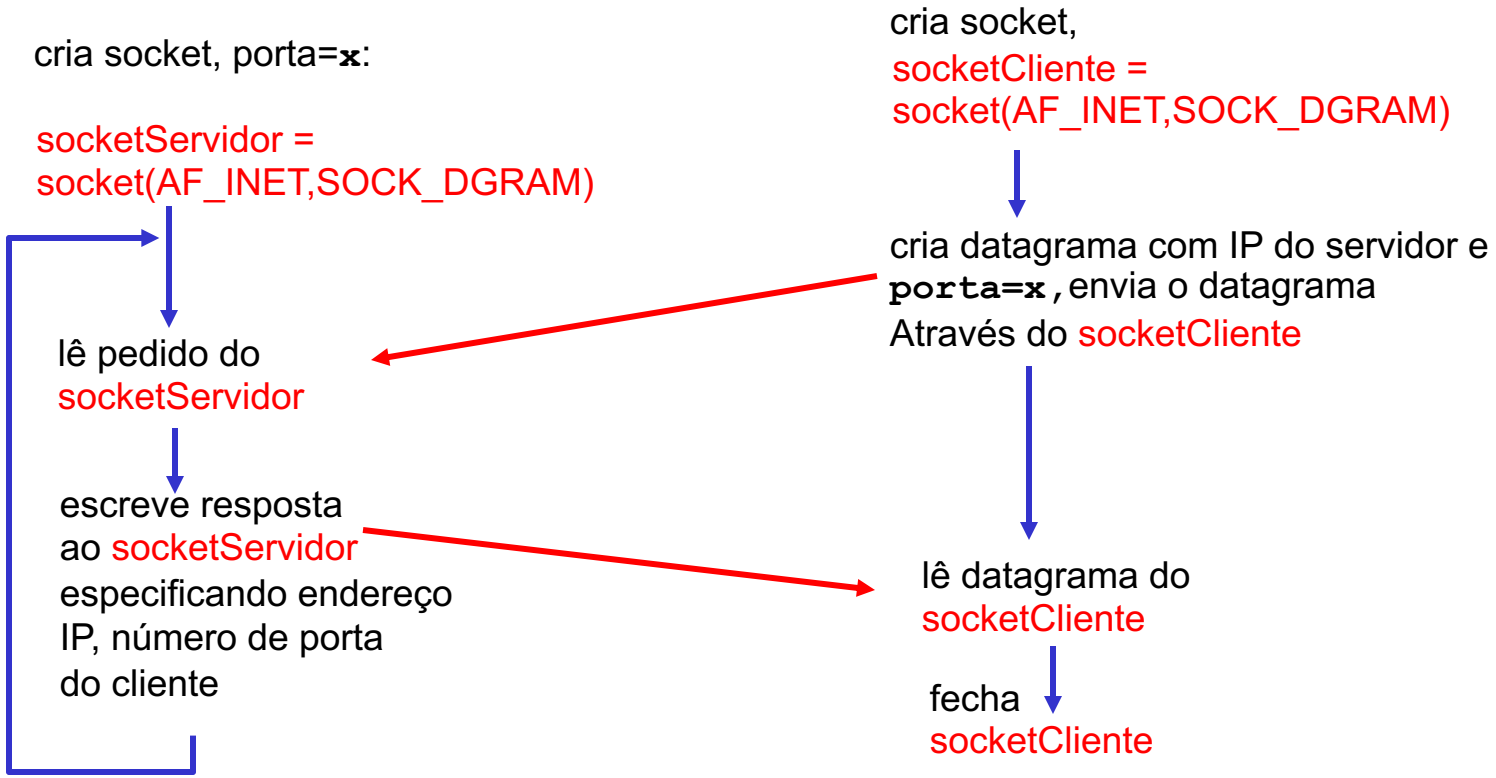
Ponto de vista da aplicação

- UDP provê transferência não confiável de grupos de bytes ("datagramas") entre cliente e servidor

Interações cliente/servidor usando o UDP

Servidor (executa em **`nomeHosp`**)

Cliente



Exemplo: cliente Python (UDP)

inclui a biblioteca de sockets do Python

cria socket UDP para servidor

obtém entrada do teclado do usuário

acrescenta o nome do servidor e número da porta à mensagem; envia pelo socket

lê caracteres de resposta do socket e converte em *string*

imprime *string* recebido e fecha socket

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET,
                      socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

Exemplo: servidor UDP

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

cria socket UDP

liga socket à porta local número 12000

loop infinito

lê mensagem do socket UDP, obtendo endereço do cliente (IP e porta do cliente)

retorna *string* em maiúsculas para este cliente

Programação com sockets usando TCP

Cliente deve contactar servidor

- processo servidor deve antes estar em execução
- servidor deve antes ter criado socket (porta) que aguarda contato do cliente

Cliente contacta servidor para:

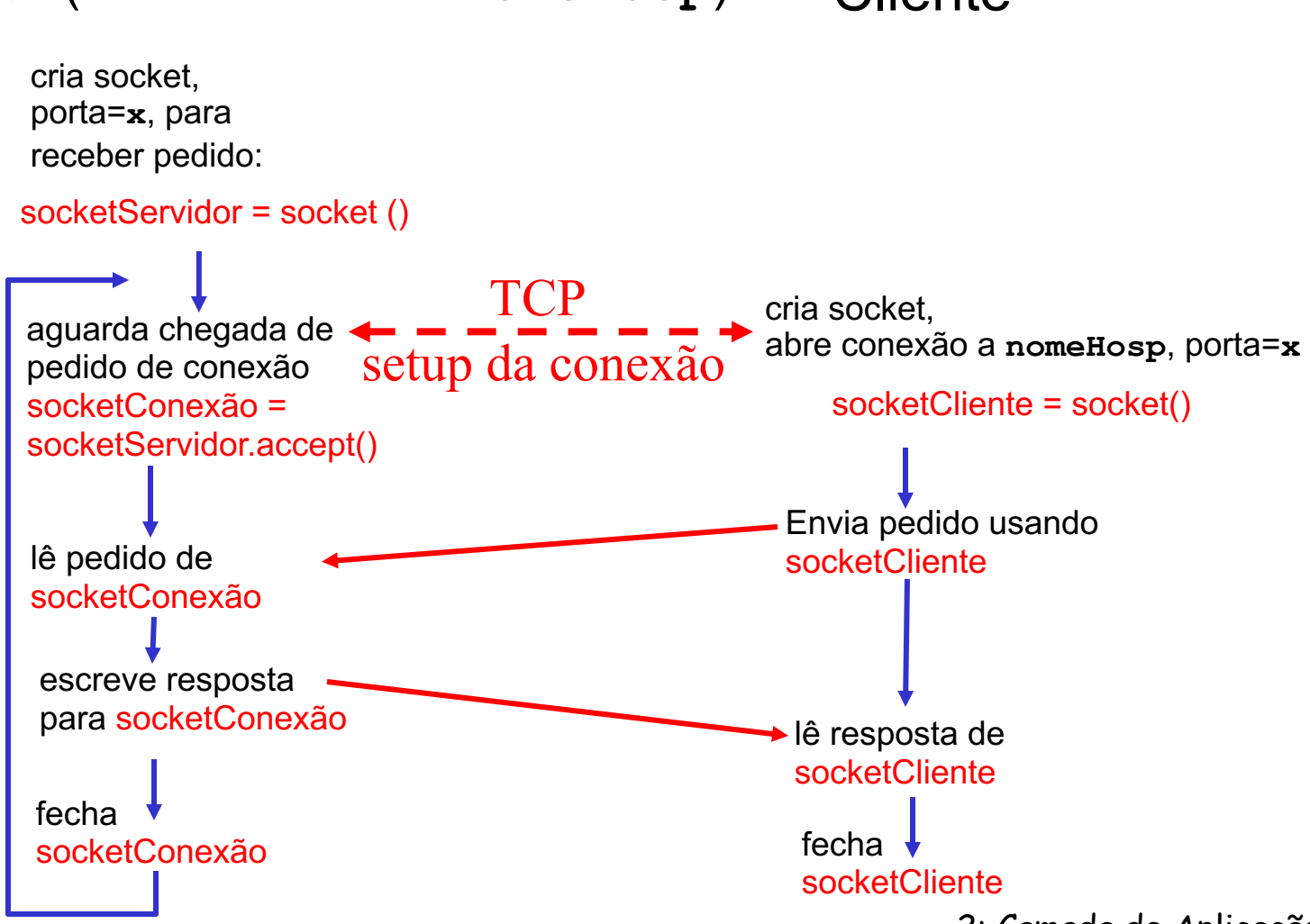
- criar socket TCP local ao cliente, especificando endereço IP, número de porta do processo servidor
- quando cliente cria socket: TCP cliente cria conexão com TCP do servidor

- Quando contatado pelo cliente, o TCP do servidor cria um novo socket para que o processo servidor possa se comunicar com o cliente
  - permite que o servidor converse com múltiplos clientes
  - Endereço IP e porta origem são usados para distinguir os clientes (mais no cap. 3)

ponto de vista da aplicação

TCP provê transferência confiável, ordenada de bytes ("tubo") entre cliente e servidor

Interações cliente/servidor usando o TCP



Exemplo: cliente TCP

inclui a biblioteca de sockets do Python

cria socket TCP socket para o servidor, porta remota 12000

não há necessidade de especificar nem o nome do servidor nem a porta

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

Exemplo: servidor TCP

cria socket TCP de recepção

servidor inicia a escuta por solicitações TCP

loop infinito

servidor espera no accept() por solicitações, um novo socket é criado no retorno

lê bytes do socket (mas não precisa ler endereço como no UDP)

fecha conexão para este cliente (mas *não* o socket de recepção)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                                encode())

    connectionSocket.close()
```

Capítulo 2: Resumo

Nosso estudo sobre aplicações de rede está agora completo!

- Arquiteturas de aplicações
  - cliente-servidor
  - P2P
- Requisitos de serviço das aplicações:
  - confiabilidade, banda, atraso
- Modelos de serviço de transporte da Internet
  - orientado à conexão, confiável: TCP
  - não confiável, datagramas: UDP
- Protocolos específicos:
  - HTTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent
  - fluxos de vídeo, CDNs
- programação de sockets: sockets UDP e TCP

Capítulo 2: Resumo

Mais importante: aprendemos sobre *protocolos*

- troca típica de mensagens pedido/resposta
  - cliente solicita info ou serviço
  - servidor responde com dados, código de *status*
- formatos de mensagens:
  - cabeçalhos: campos com info sobre dados (metadados)
  - dados: info (carga) sendo comunicada

Temas importantes:

- msgs de controle vs. dados
  - na banda, fora da banda
- centralizado vs. descentralizado
- s/ estado vs. c/ estado
- transferência de msgs confiável vs. não confiável
- "complexidade na borda da rede"