

Capítulo 2: Camada de Aplicação

charles.garrocho@ifmg.edu.br
garrocho.github.io

Baseado nos slides de Kurose e Ross

Capítulo 2: Roteiro

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

Capítulo 2: Camada de Aplicação

Metas do capítulo:

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

Algumas aplicações de rede

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

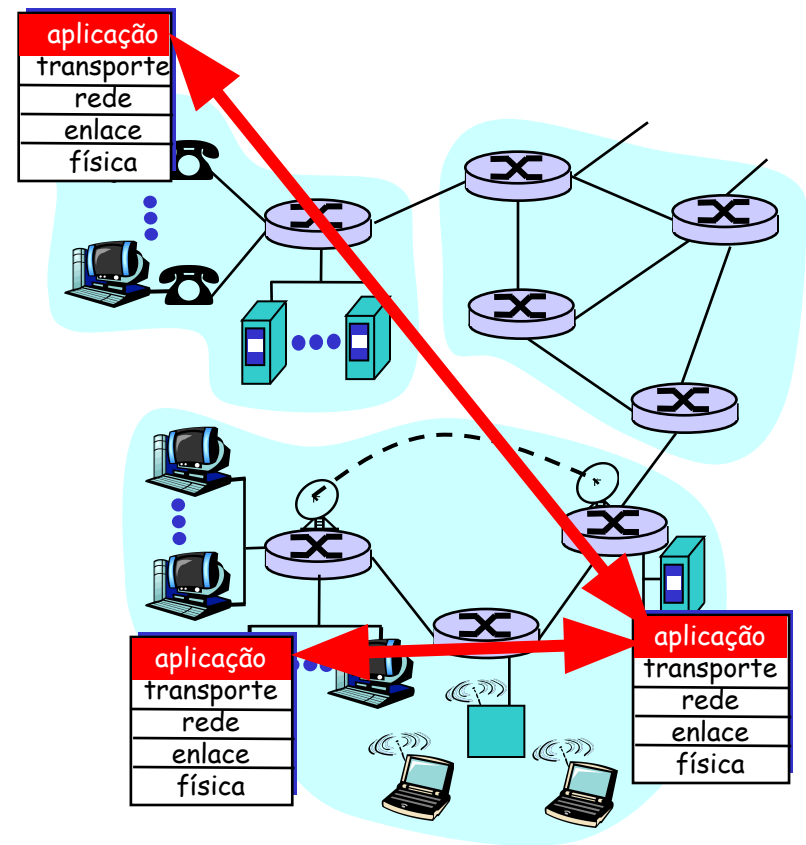
Criando uma aplicação de rede

Programas que

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

Programas não relacionados ao núcleo da rede

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



Arquiteturas das aplicações de rede

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

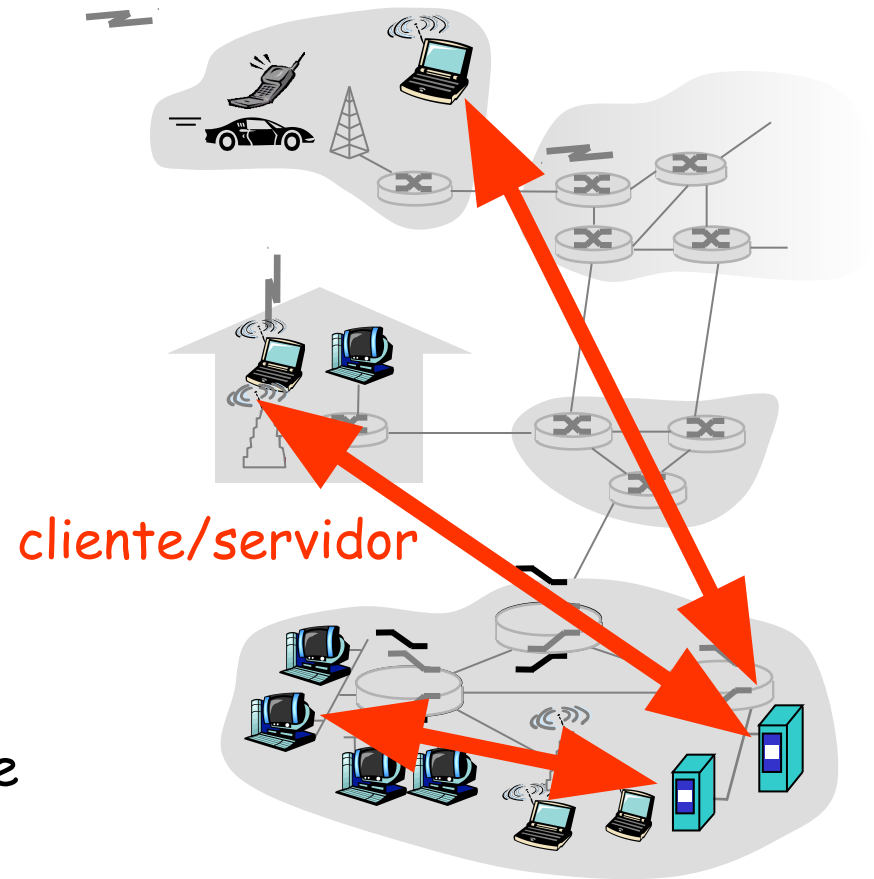
Arquitetura cliente-servidor

Servidor:

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

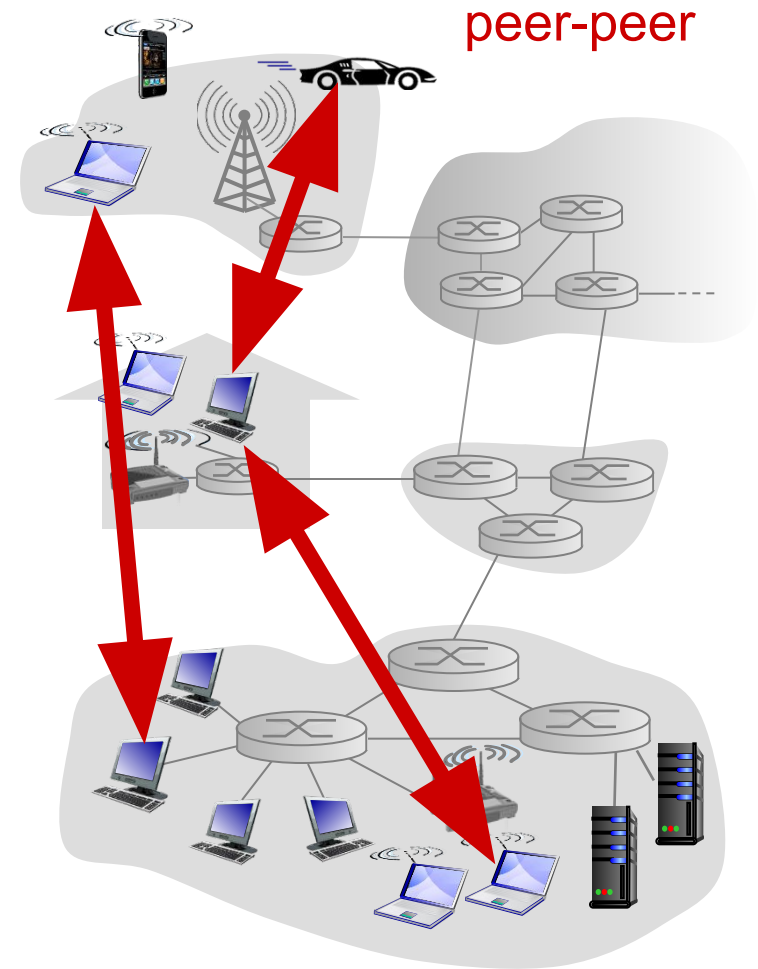
Clientes:

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



Arquitetura P2P

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



Comunicação entre Processos

- Processo:** programa que executa num sistema final
- r processos no mesmo sistema final se comunicam usando **comunicação entre processos** (definida pelo sistema operacional)
 - r processos em sistemas finais distintos se comunicam trocando **mensagens** através da rede

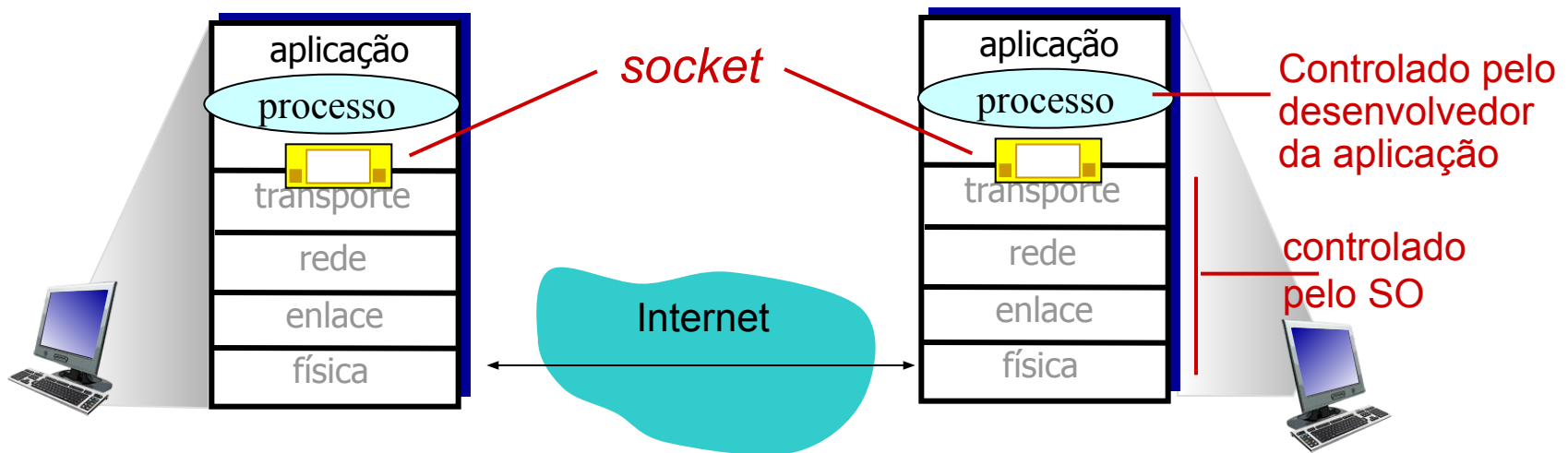
Processo cliente:
processo que inicia a comunicação

Processo servidor:
processo que espera ser contatado

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

Sockets

- r Os processos enviam/ recebem mensagens para/dos seus *sockets*
- r Um socket é análogo a uma porta
 - m Processo transmissor envia a mensagem através da porta
 - m 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

- r Para que um processo receba mensagens, ele deve possuir um **identificador**
- r Cada hospedeiro possui um **endereço IP** único de 32 bits
- r **P:** o endereço IP do hospedeiro no qual o processo está sendo executado é suficiente para identificar o processo?
- r **Resposta:** Não, muitos processos podem estar executando no mesmo hospedeiro
- r O identificador inclui tanto o **endereço IP** quanto os **números das portas** associadas com o processo no hospedeiro .
- r Exemplo de números de portas:
 - m Servidor HTTP: 80
 - m Servidor de Correio: 25
- r Para enviar uma msg HTTP para o servidor Web gaia.cs.umass.edu
 - m **Endereço IP:** 128.119.245.12
 - m **Número da porta:** 80
- r Mais sobre isto posteriormente.

De que serviços uma aplicação necessita?

Integridade dos dados (sensibilidade a perdas)

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

Temporização (sensibilidade a atrasos)

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

Vazão (throughput)

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

Segurança

- r 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:

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

Serviço UDP:

- r transferência de dados não confiável* entre processos remetente e receptor
 - r 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

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

SSL

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

SSL está na camada de aplicação

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

API do socket SSL

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

Os protocolos da camada de aplicação definem

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

Protocolos abertos:

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

Protocolos proprietários:

- r Ex., Skype

Exercícios

1. Em uma troca de mensagens de whatsapp entre dois usuários, identifique os processos cliente e servidor.
2. Você foi contratado para alterar o funcionamento de uma aplicação de rede que atualmente está congestionando a rede de comunicação da empresa. A aplicação atual utiliza TCP ou UDP? Para qual protocolo você migraria ou permaneceria? Explique!
3. Que tipo de serviços uma aplicação de rede como a NetFlix necessita? Explique!

Para casa: Faça as listas de exercícios no site/moodle.

Capítulo 2: Roteiro

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

A Web e o HTTP

Primeiro uma revisão...

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

www.someschool.edu / someDept/pic.gif

nome do hospedeiro nome do caminho

Protocolo HTTP

HTTP: *hypertext transfer protocol*

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



Mais sobre o protocolo HTTP

Usa serviço de transporte TCP:

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

HTTP é "sem estado"

- r servidor não mantém informação sobre pedidos anteriores do cliente

Nota
Protocolos que mantêm "estado" são complexos!

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

Conexões HTTP

HTTP não persistente

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

HTTP persistente

- r 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/algumDepartamento/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



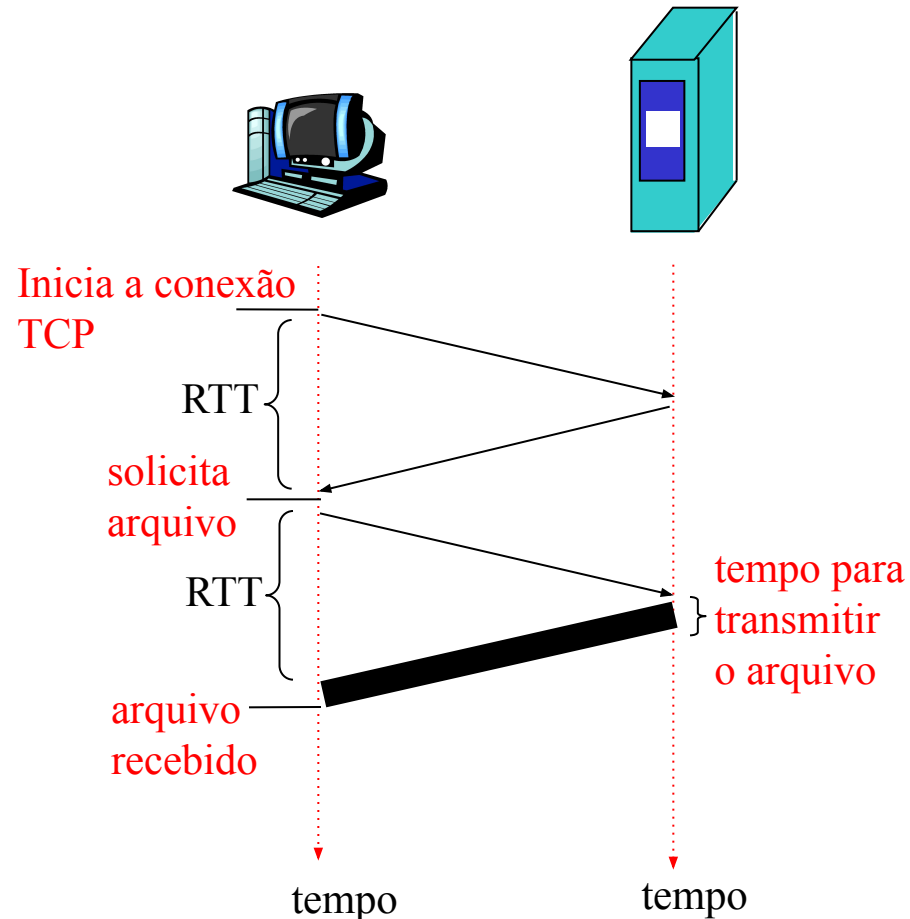
Modelagem do tempo de resposta

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:

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

total = $2RTT + \text{tempo de transmissão do arquivo}$



HTTP persistente

Problemas com o HTTP não persistente:

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

HTTP persistente

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

Mensagem de requisição HTTP

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

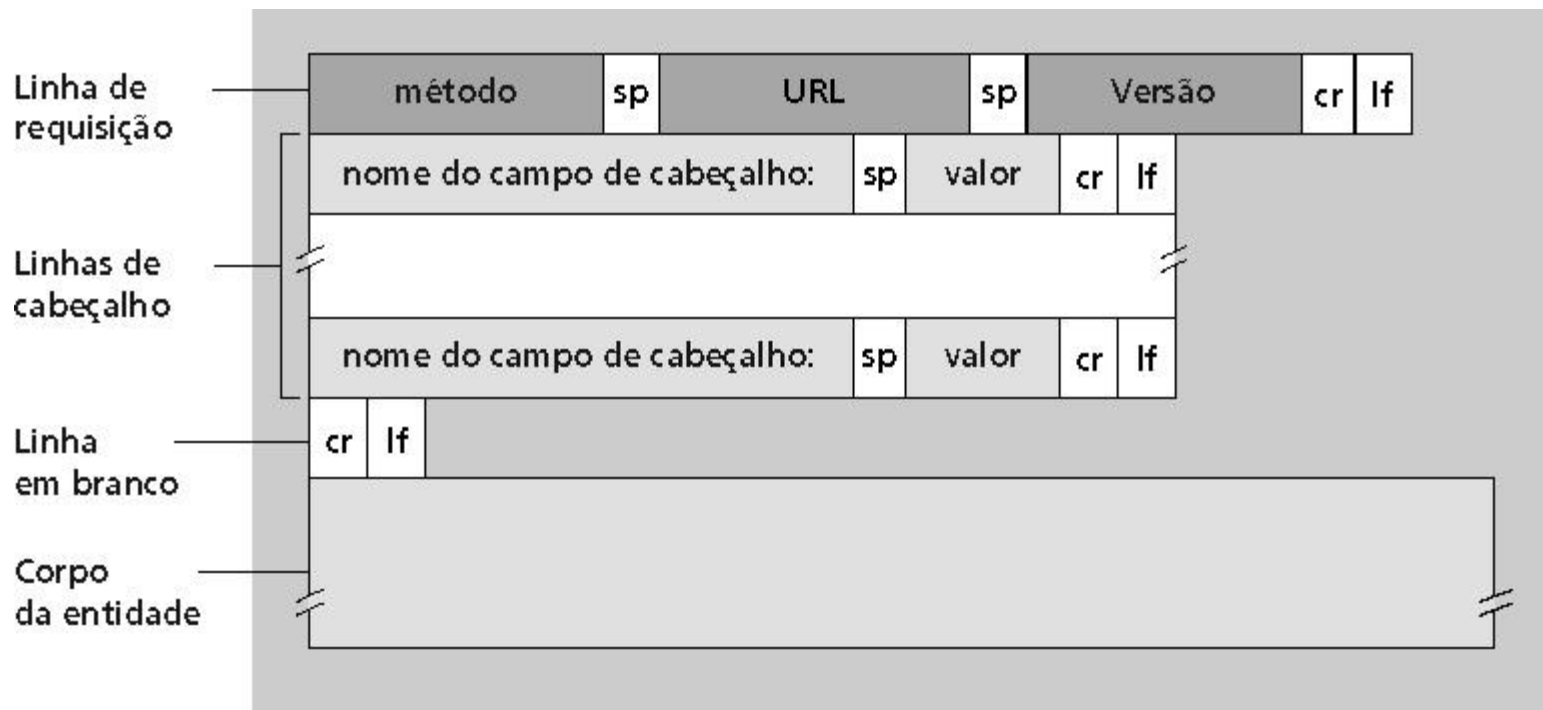
linha da requisição
(comandos GET,
POST, HEAD)

linhas de
cabeçalho

Carriage return,
line feed
indicam fim
de mensagem

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept:
    text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Mensagem de requisição HTTP: formato geral



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

Enviando conteúdo de formulário

Método POST:

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

Método URL:

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

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

Tipos de métodos

HTTP/1.0

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

HTTP/1.1

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

Mensagem de resposta HTTP

linha de status

(protocolo,
código de status,
frase de status)

linhas de
cabeçalho

dados, p.ex.,
arquivo html
solicitado

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


códigos de status da resposta HTTP

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

200 OK

m sucesso, objeto pedido segue mais adiante nesta mensagem

301 Moved Permanently

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

400 Bad Request

m mensagem de pedido não entendida pelo servidor

404 Not Found

m documento pedido não se encontra neste servidor

505 HTTP Version Not Supported

m versão de http do pedido não usada por este servidor

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

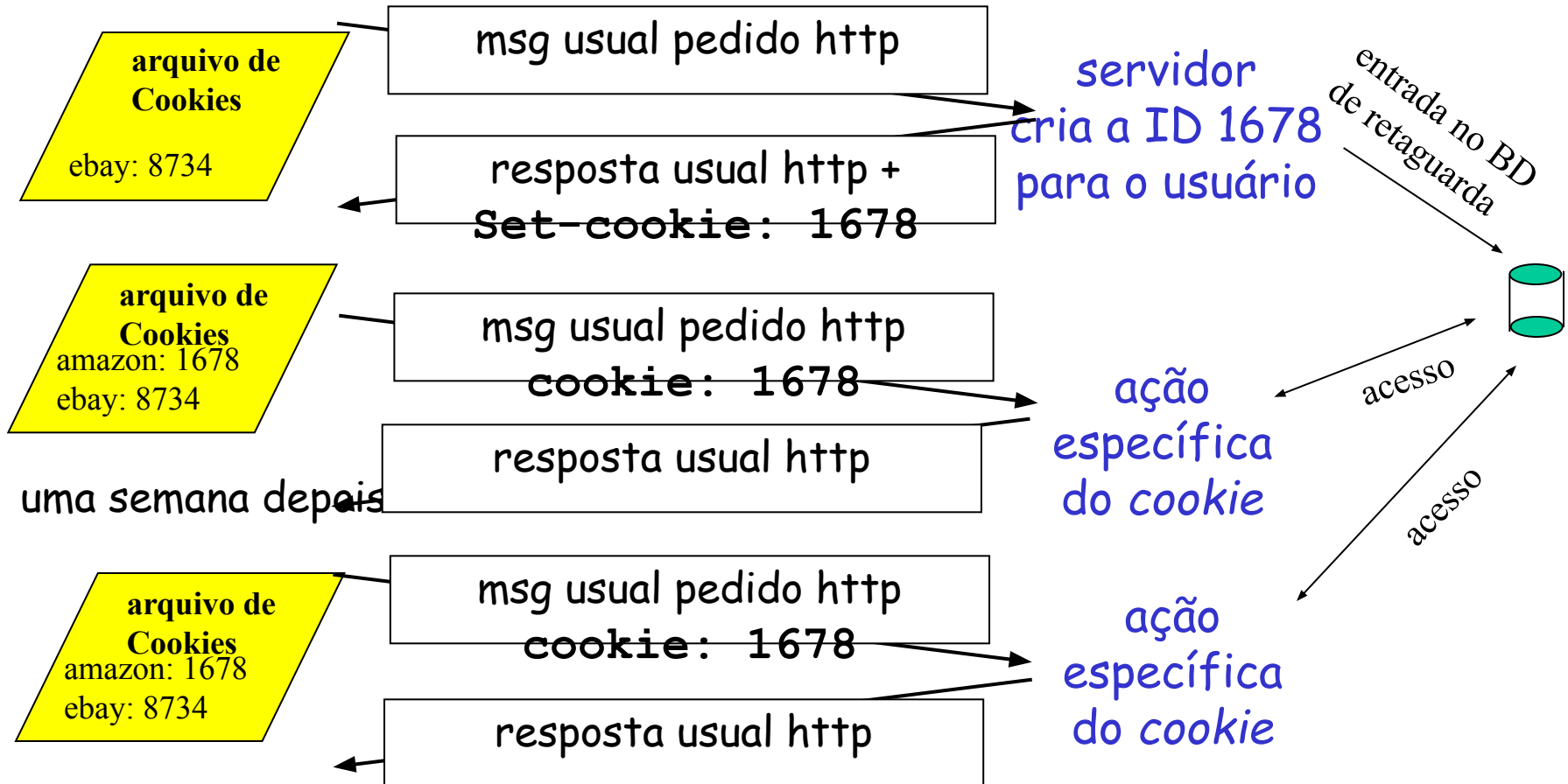
Exemplo:

- m Suzana acessa a Internet sempre do mesmo PC
- m Ela visita um sítio específico de comércio eletrônico pela primeira vez
- m 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.)

cliente

servidor



Cookies (continuação)

O que os cookies podem obter:

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

nota

Cookies e privacidade:

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

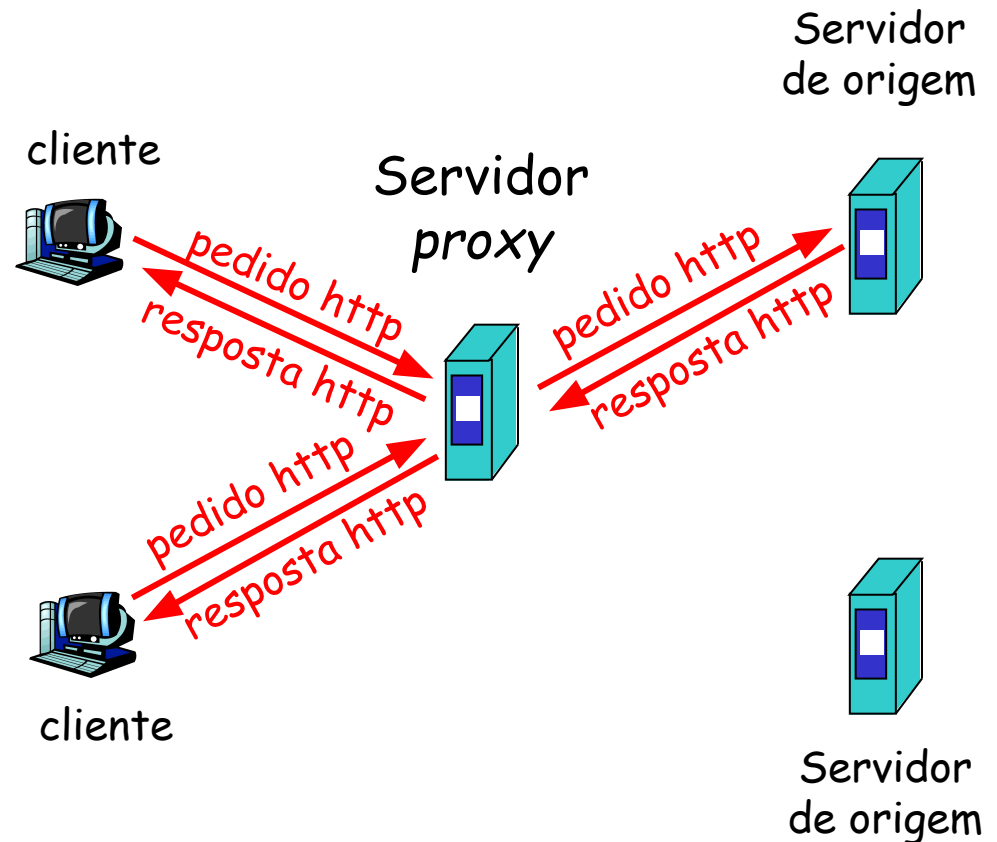
Como manter o "estado":

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

Cache Web (servidor proxy)

Meta: atender pedido do cliente sem envolver servidor de origem

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



Mais sobre Caches Web

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

Para que fazer cache Web?

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

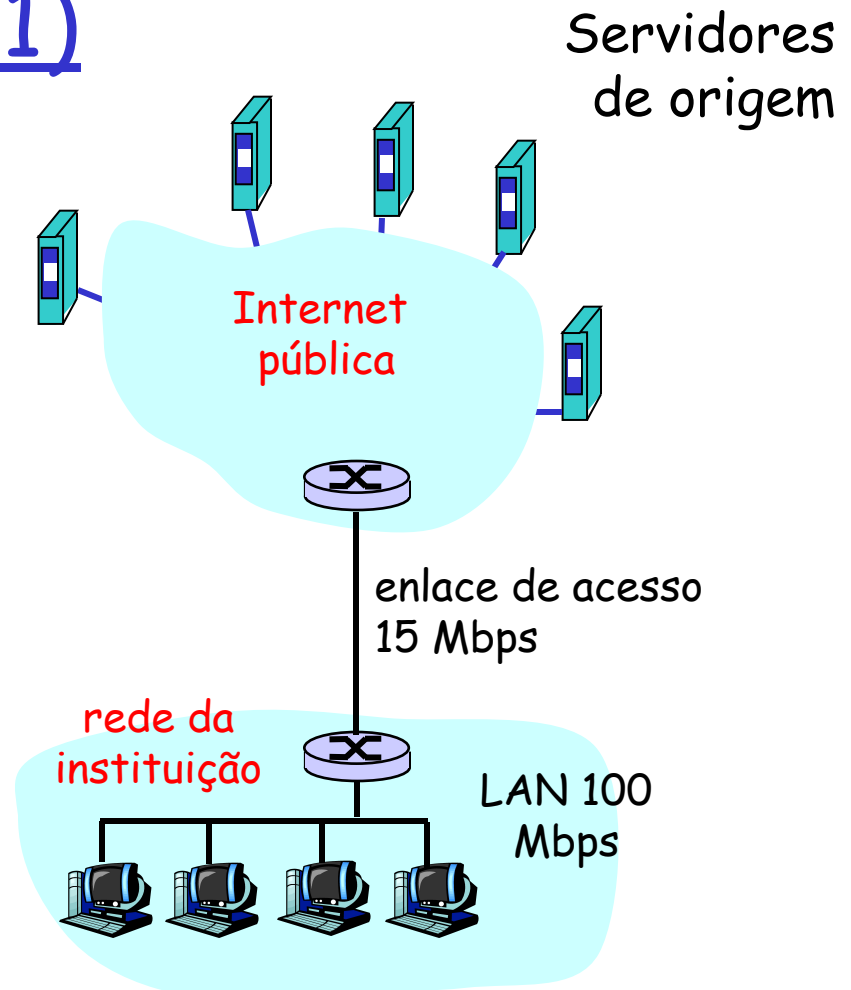
Exemplo de cache (1)

Hipóteses

- r Tamanho médio de um objeto = 1 Mbits
- r Taxa média de solicitações dos *browsers* de uma instituição para os servidores originais = 15/seg
- r Atraso do roteador institucional para qualquer servidor origem e de volta ao roteador = 2seg

Consequências

- r Utilização da LAN = 0,15%
- r Utilização do canal de acesso = 99% *problema!*



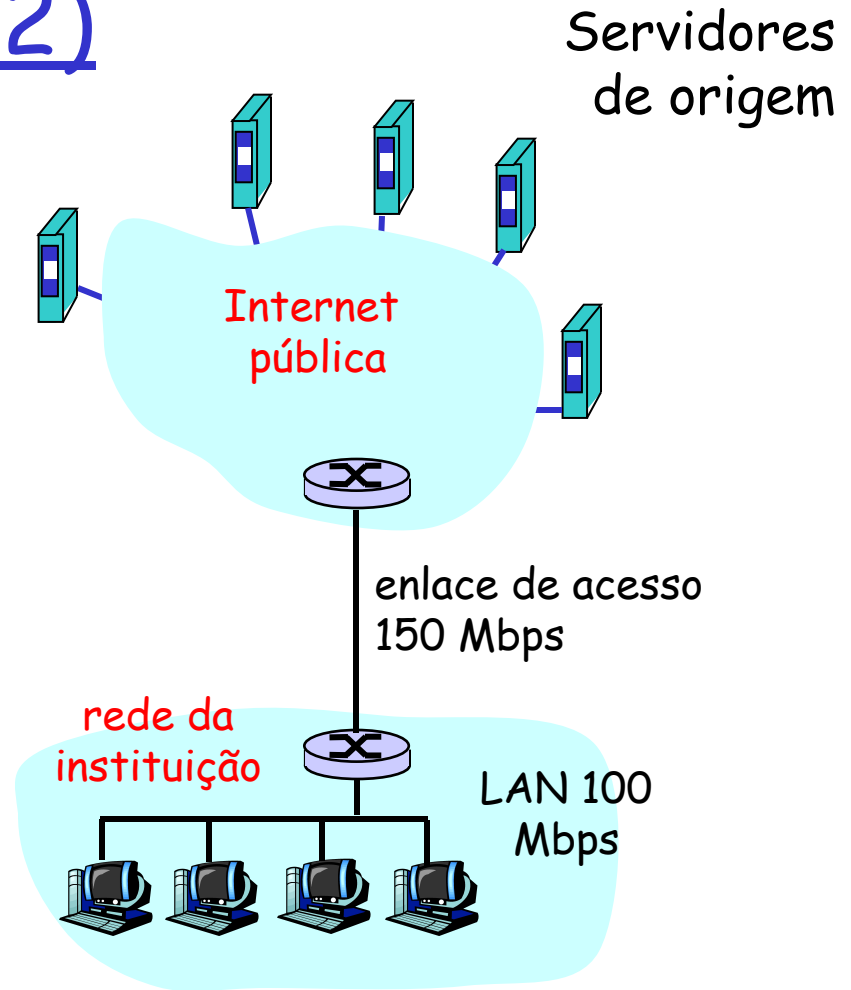
Exemplo de cache (2)

Solução em potencial

- r Aumento da largura de banda do canal de acesso para, por exemplo, 150 Mbps

Consequências

- r Utilização da LAN = 0,15%
- r Utilização do canal de acesso = 9,9%
- r Frequentemente este é uma ampliação cara



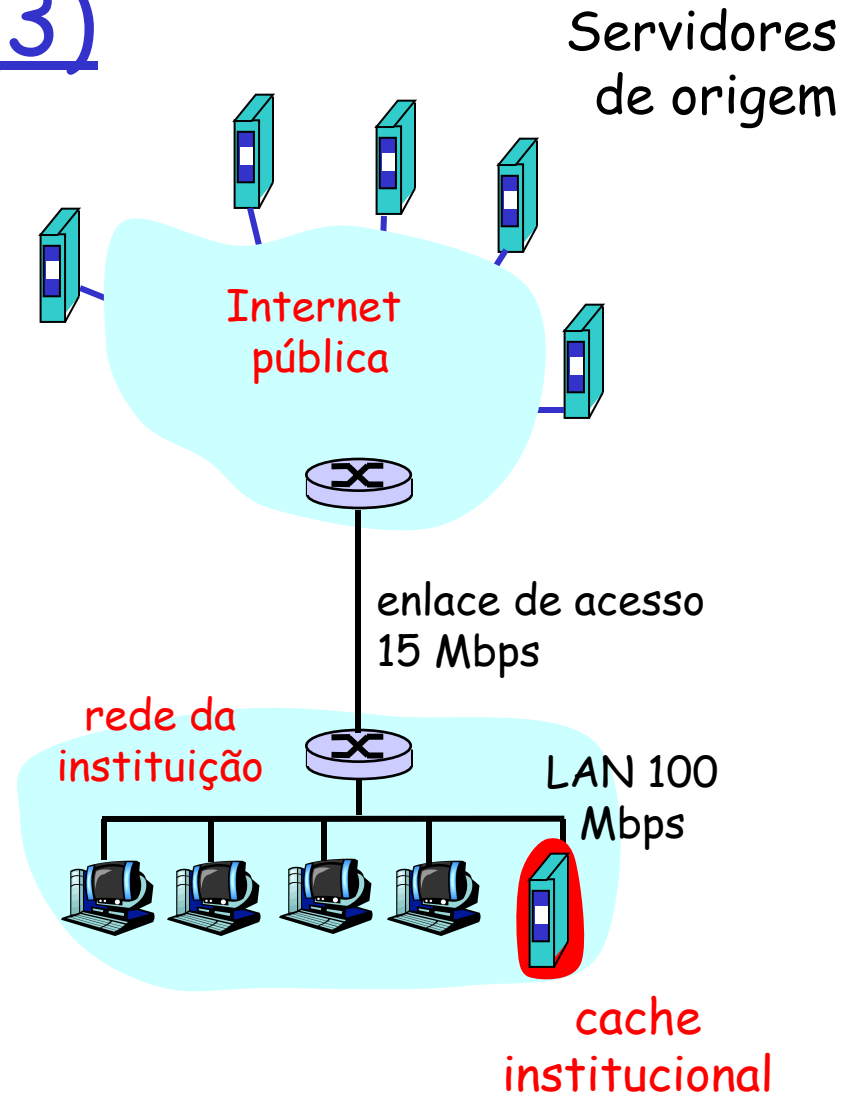
Exemplo de cache (3)

Instale uma cache

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

Consequências

- r 40% dos pedidos serão atendidos quase que imediatamente
- r 60% dos pedidos serão servidos pelos servidores de origem
- r Utilização do canal de acesso é reduzido para 60%, resultando em atrasos desprezíveis (ex. 10 mseg)



GET condicional

r **Meta:** não enviar objeto se cliente já tem (no cache) versão atual

m Sem atraso para transmissão do objeto

m Diminui a utilização do enlace

r cache: especifica data da cópia no cache no pedido HTTP

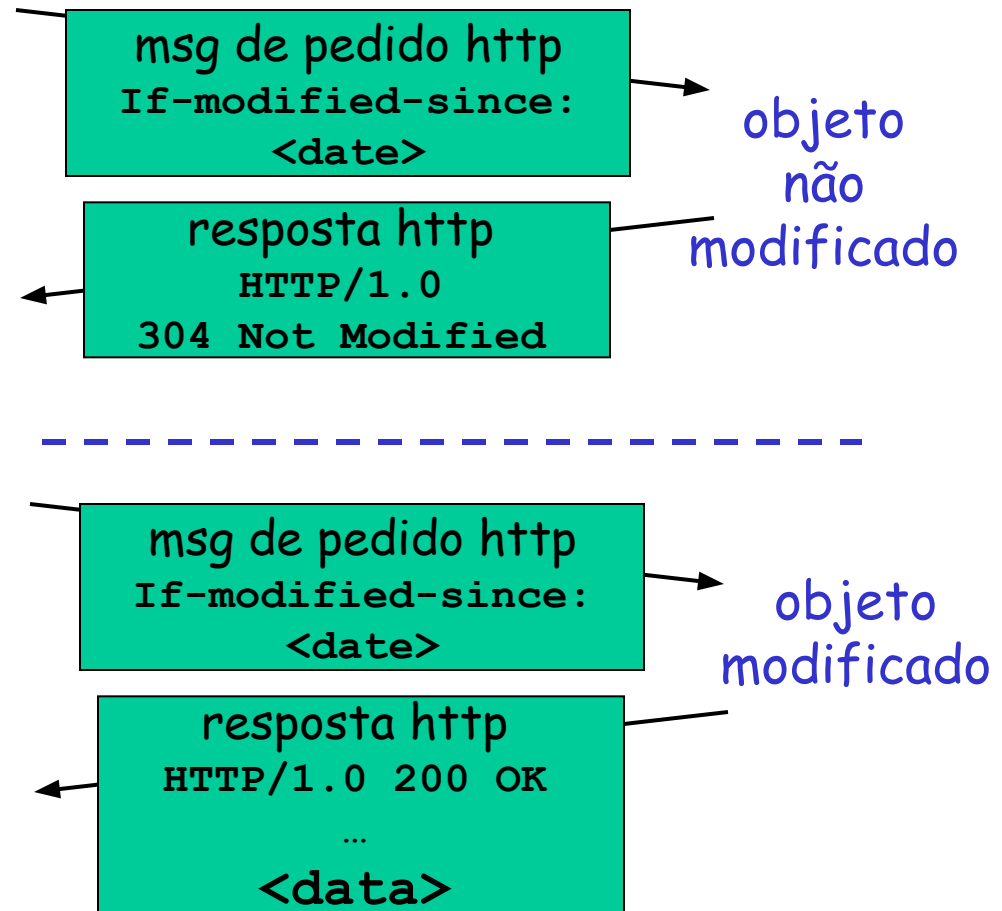
If-modified-since:
<date>

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

HTTP/1.0 304 Not
Modified

cache

servidor



Capítulo 2: Roteiro

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

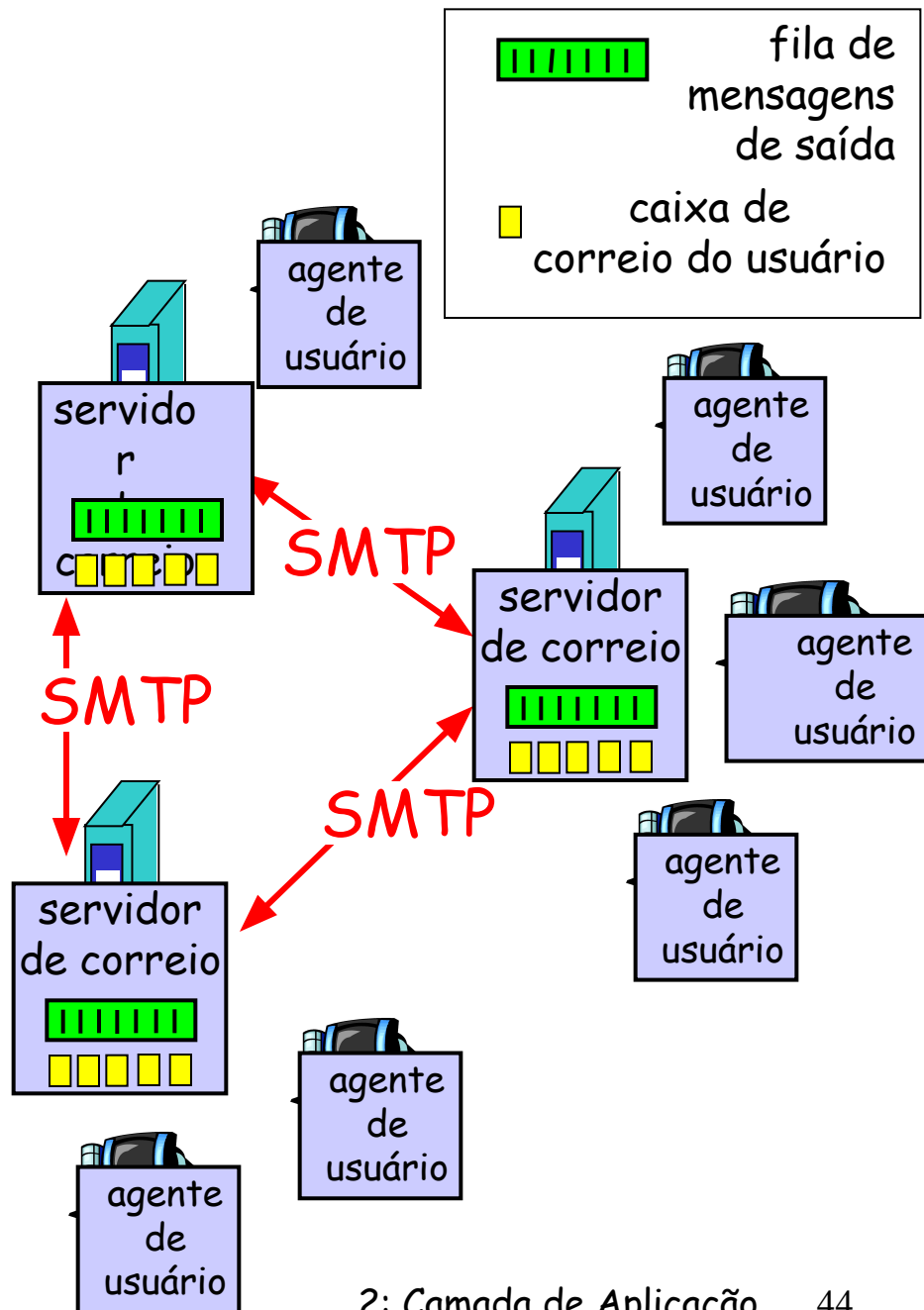
Correio Eletrônico

Três grandes componentes:

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

Agente de Usuário

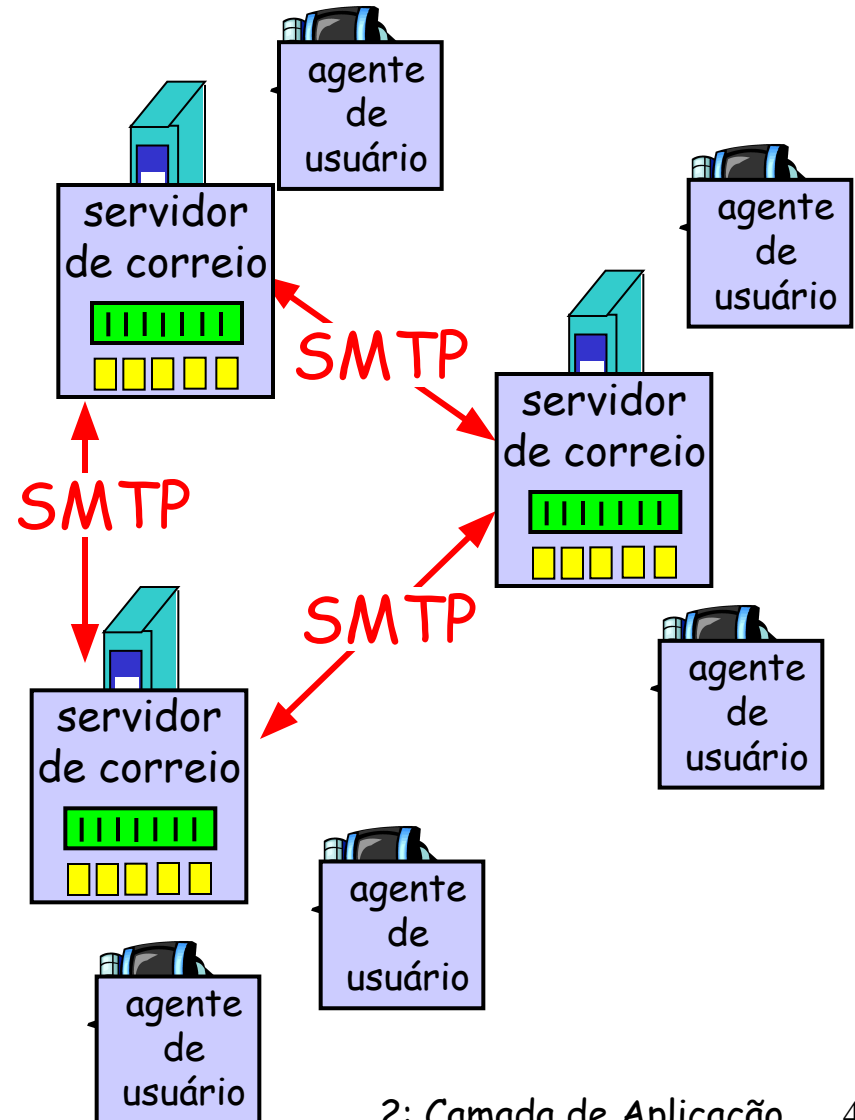
- r a.k.a. "leitor de correio"
- r compor, editar, ler mensagens de correio
- r p.ex., Outlook, Thunderbird, cliente de mail do iPhone
- r mensagens de saída e chegando são armazenadas no servidor



Correio Eletrônico: servidores de correio

Servidores de correio

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



Correio Eletrônico: SMTP [RFC 2821]

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

Gerência da Porta 25

Configure a
porta de envio
de suas mensagens para

587!

Com a Gerência da Porta 25, o Brasil vai reduzir
o volume de spams enviados em nosso país.

Você ajuda o Brasil a melhorar a Internet e
ainda evita dores de cabeça.

Conheça neste site mais detalhes do
Gerenciamento da Porta 25.

Afinal, quem tem que ficar de fora são os spams,
e não você!

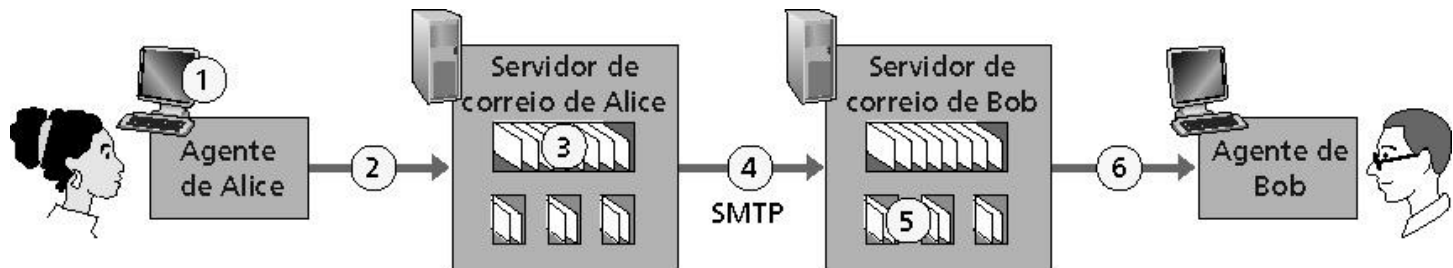
Feche a porta para os spams!



<http://antispam.br/>

Cenário: Alice envia uma msg para Bob

- 1) Alice usa o UA para compor uma mensagem "para" bob@school.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



Legenda:



Fila de mensagens



Caixa postal do usuário

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:

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

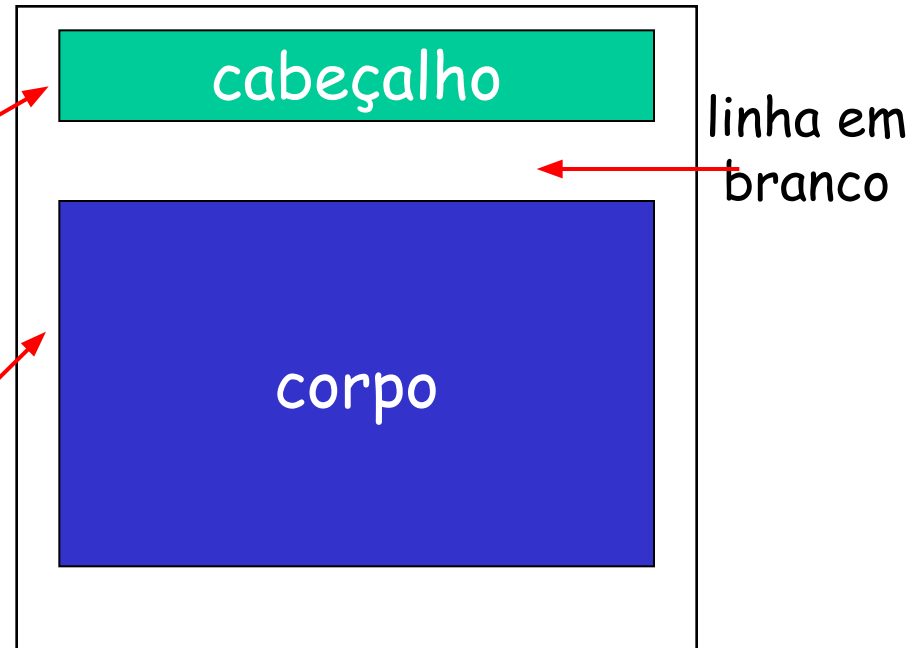
estes comandos permitem que você envie correio sem usar um cliente (leitor de correio)

Formato de uma mensagem

SMTP: protocolo para trocar
msgs de correio

RFC 822: padrão para formato de
mensagem de texto:

- r linhas de cabeçalho, p.ex.,
 - m To:
 - m From:
 - m Subject:*diferentes* dos comandos de
smtp FROM, RCPT TO
- r corpo
 - m a "mensagem", somente de
caracteres ASCII



Formato de uma mensagem: extensões para multimídia

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

versão MIME

método usado
p/ codificar dados

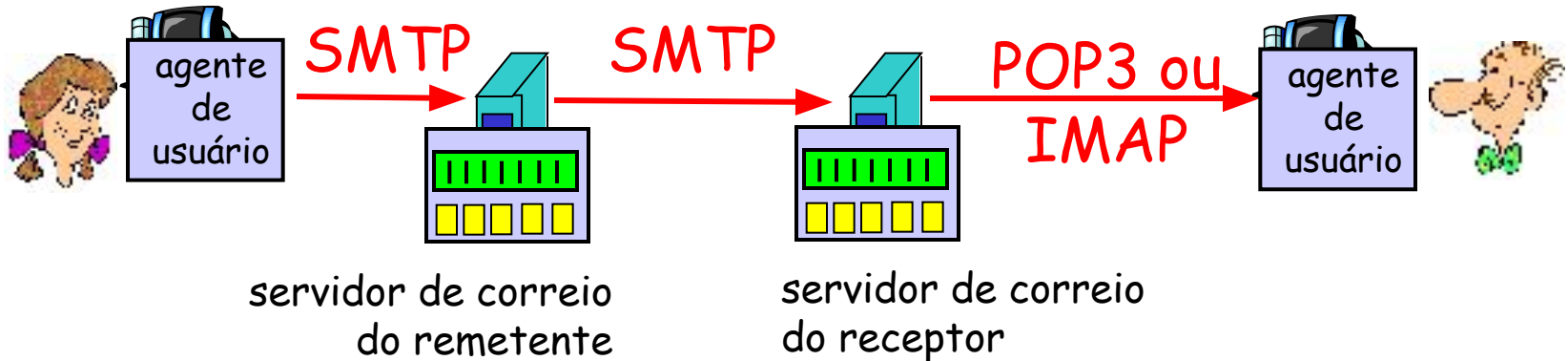
tipo, subtipo de
dados multimídia,
declaração parâmetros

Dados codificados

```
From: ana@consumidor.br
To: bernardo@doces.br
Subject: Imagem de uma bela torta
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.....
.....base64 encoded data
```

Protocolos de acesso ao correio



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

Protocolo POP3

fase de autorização

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

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

fase de transação, cliente:

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

```
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

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

IMAP

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

Capítulo 2: Roteiro

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

DNS: Domain Name System

Pessoas: muitos identificadores:

- m CPF, nome, no. da Identidade

hospedeiros, roteadores Internet :

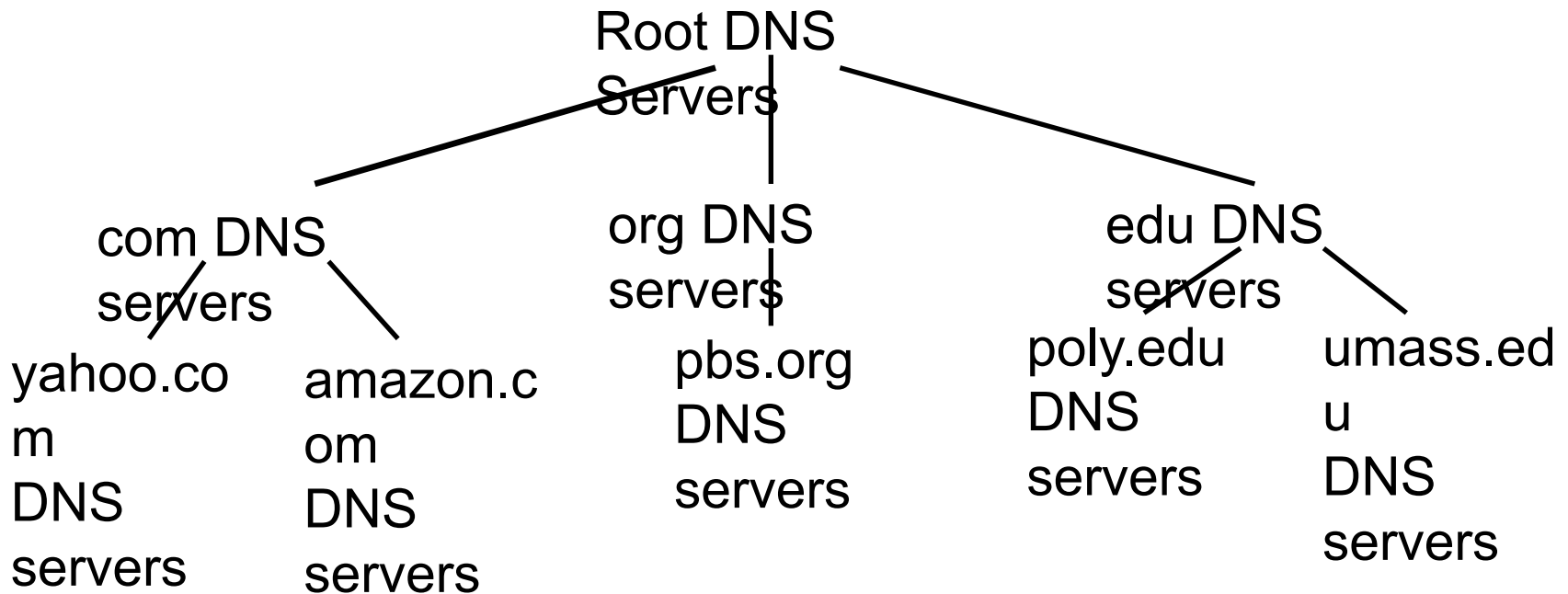
- m endereço IP (32 bit) - usado p/ endereçar datagramas
- m "nome", ex., www.yahoo.com - usado por gente

P: como mapear entre nome e endereço IP?

Domain Name System:

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

Base de Dados Hierárquica e Distribuída



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

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

DNS: Servidores raiz

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



13 servidores de
nome raiz em todo o
mundo

Servidores TLD e Oficiais

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

Servidor DNS Local

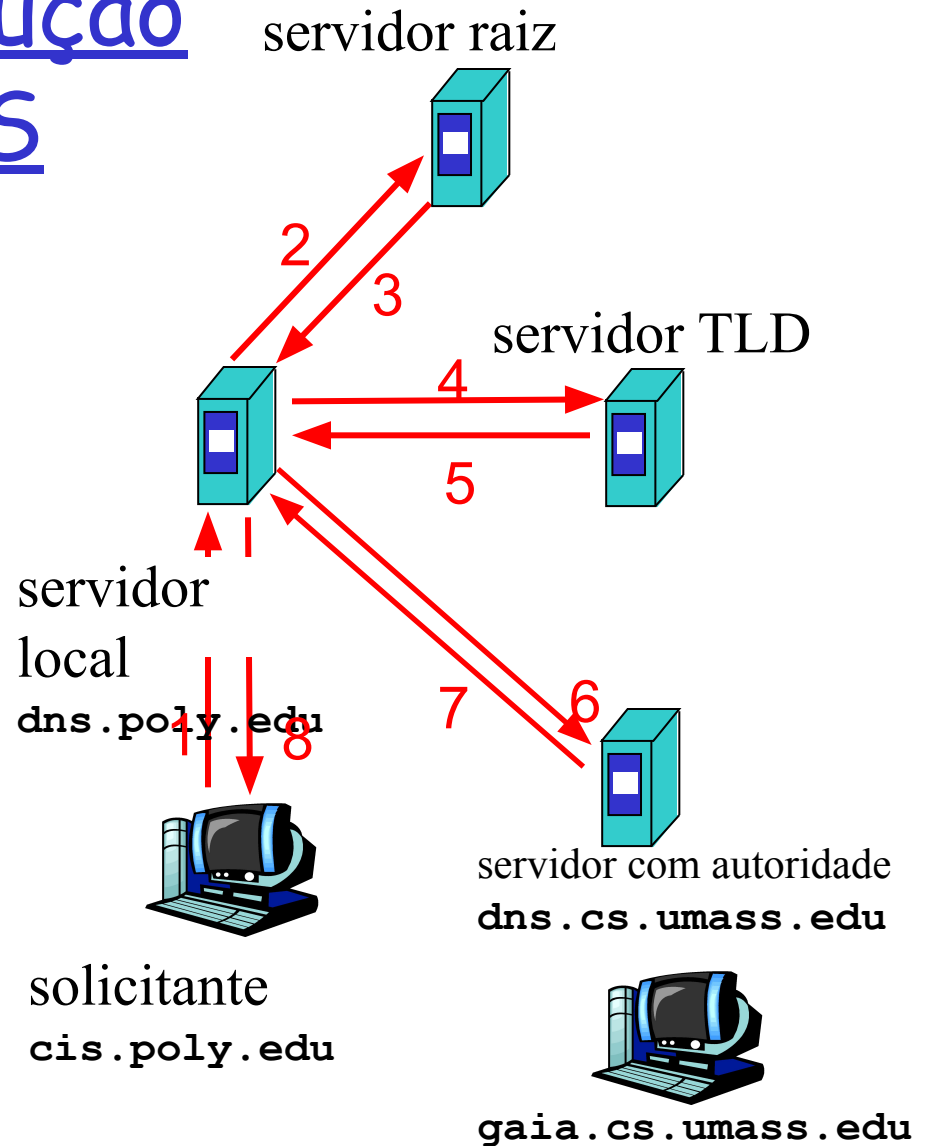
- r Não pertence necessariamente à hierarquia
- r Cada ISP (ISP residencial, companhia, universidade) possui um.
 - m Também chamada do "servidor de nomes default"
- r Quanto um hospedeiro faz uma consulta DNS, a mesma é enviada para o seu servidor DNS local
 - m Possui uma cache local com pares de tradução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

Exemplo de resolução de nome pelo DNS

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

consulta interativa:

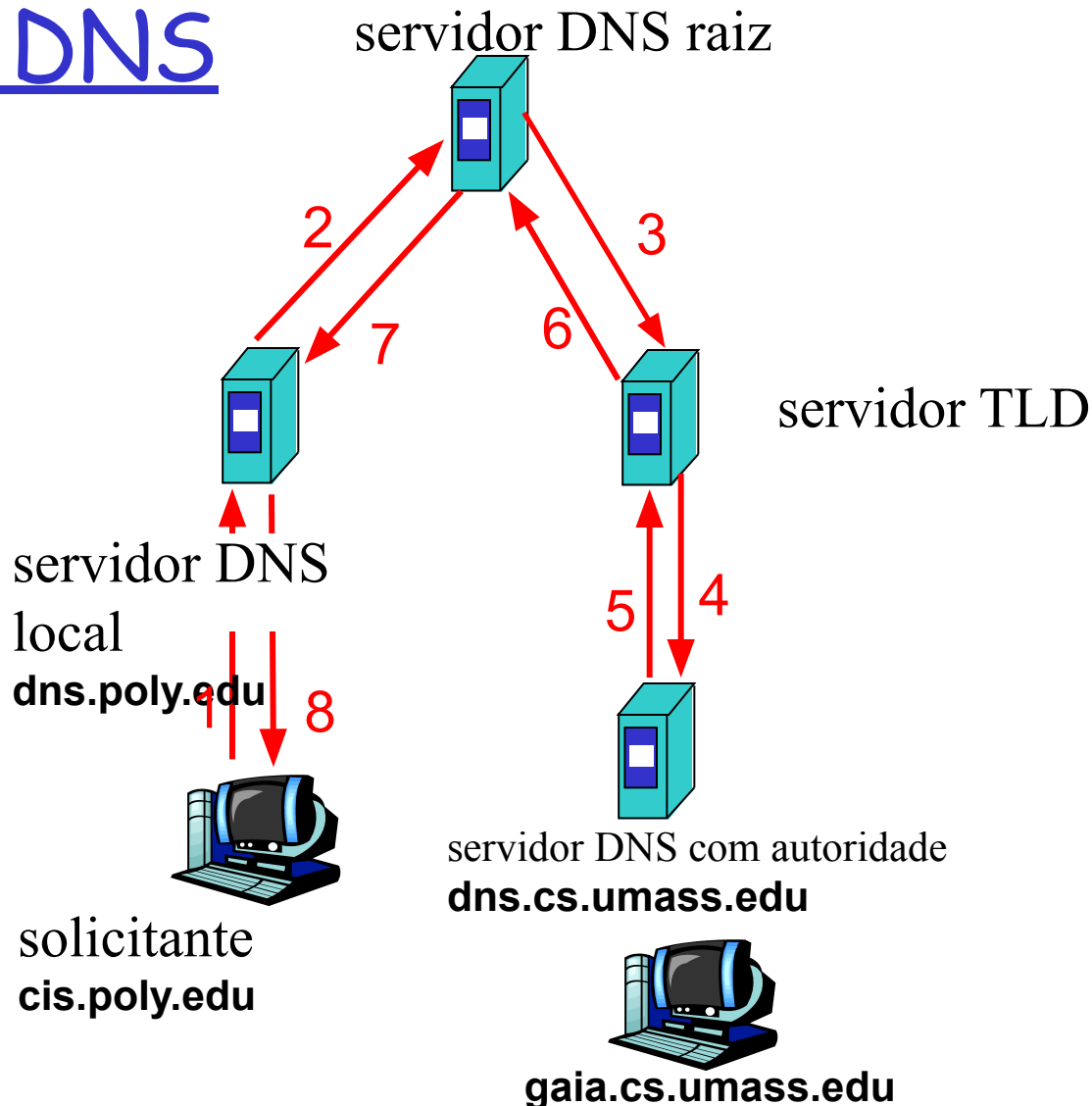
- r servidor consultado responde com o nome de um servidor de contato
- r "Não conheço este nome, mas pergunte para esse servidor"



Exemplo de resolução de nome pelo DNS

consulta recursiva:

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



DNS: uso de cache, atualização de dados

- r uma vez que um servidor qualquer aprende um mapeamento, ele o coloca numa **cache** local
 - m entradas na cache são sujeitas a temporização (desaparecem) depois de um certo tempo (TTL)
- r Entradas na cache podem estar **desatualizadas** (tradução nome/endereço do tipo melhor esforço!)
 - m 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
- r mecanismos de atualização/notificação propostos na RFC 2136

Registros DNS

DNS: BD distribuído contendo *registros de recursos*

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

r Tipo=A

- m **nome** é nome de hospedeiro
- m **valor** é o seu endereço IPv4
- m Tipo=AAAA para IPv6

r Tipo=NS

- m **nome** é domínio (p.ex. foo.com.br)
- m **valor** é endereço IP de servidor oficial de nomes para este domínio

r Tipo=CNAME

- m **nome** é nome alternativo (alias) para algum nome “canônico” (verdadeiro)
- m www.ibm.com é na verdade servereast.backup2.ibm.com
- m **valor** é o nome canônico

r Tipo=MX

- m **valor** é nome do servidor de correio associado ao **nome**

Capítulo 2: Roteiro

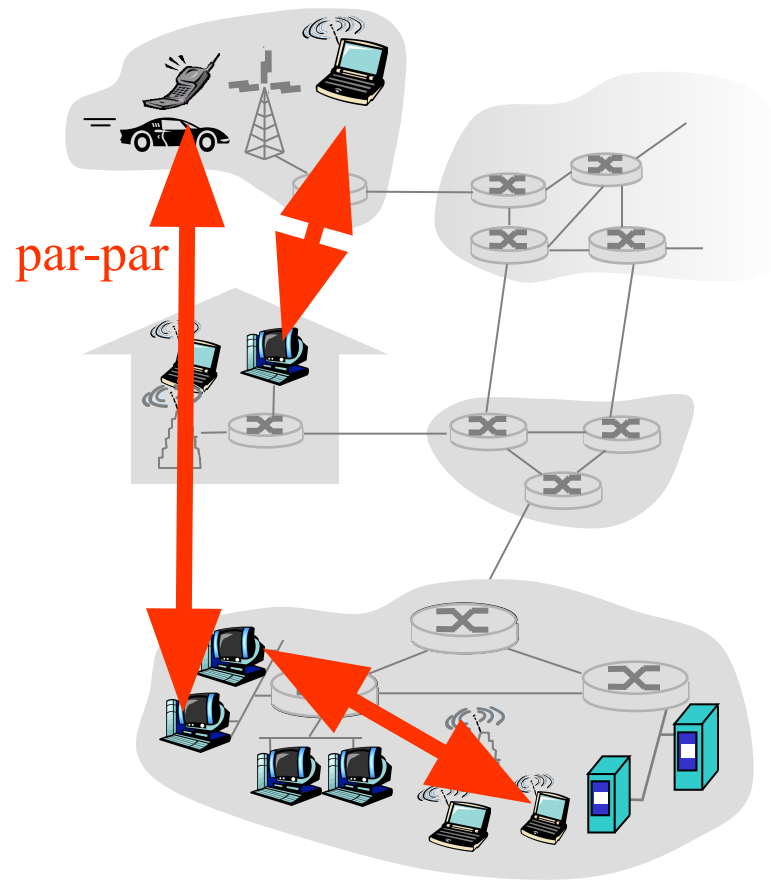
- r 2.1 Princípios de aplicações de rede
- r 2.2 A Web e o HTTP
- r 2.3 Correio Eletrônico na Internet
- r 2.4 DNS: o serviço de diretório da Internet
- r 2.5 Aplicações P2P
- r 2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)
- r 2.7 Programação de *sockets* com UDP e TCP

Arquitetura P2P pura

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

- r Exemplos:

- m Distribuição de arquivos (BitTorrent)
- m Streaming (KanKan)
- m VoIP (Skype)



Distribuição de arquivo P2P: BitTorrent

r arquivos divididos em blocos de 256kb

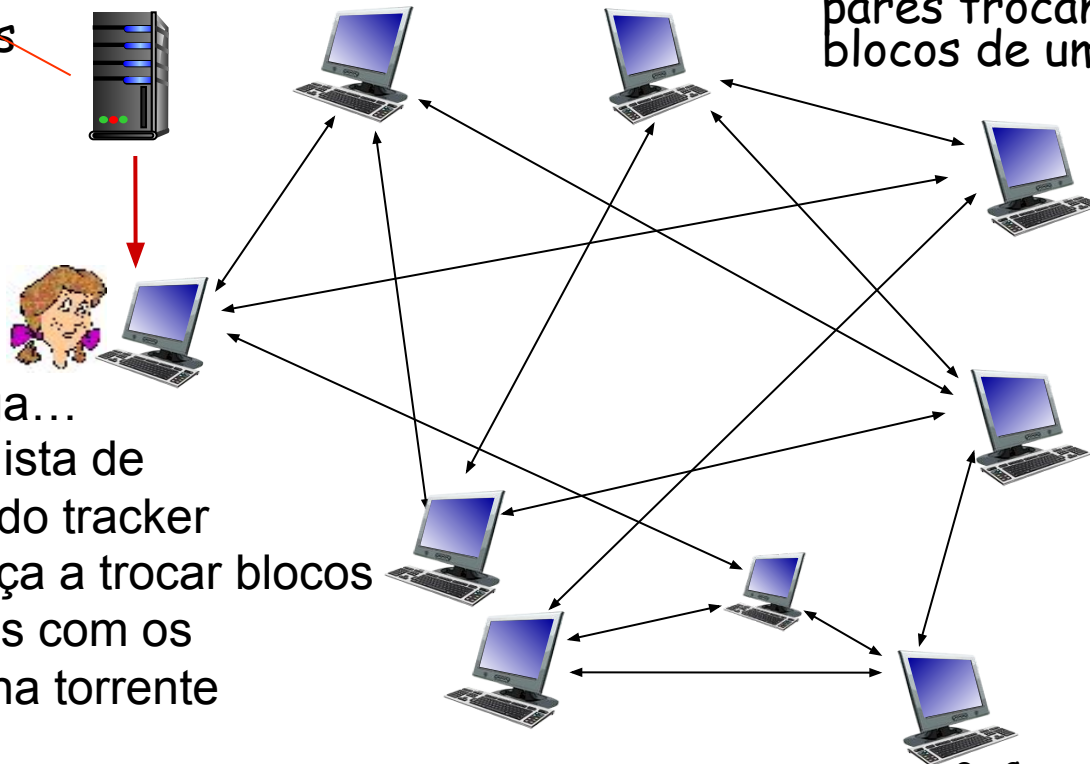
r Pares numa torrente enviam/recebem blocos do arquivo

tracker:

registra pares
participantes
de uma
torrente

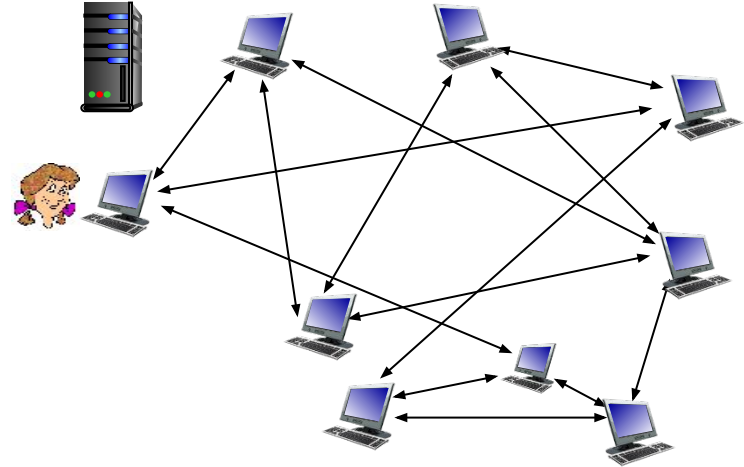
torrente: grupo de
pares trocando
blocos de um arquivo

Alice chega...
... obtém lista de
parceiros do tracker
... e começa a trocar blocos
de arquivos com os
parceiros na torrente



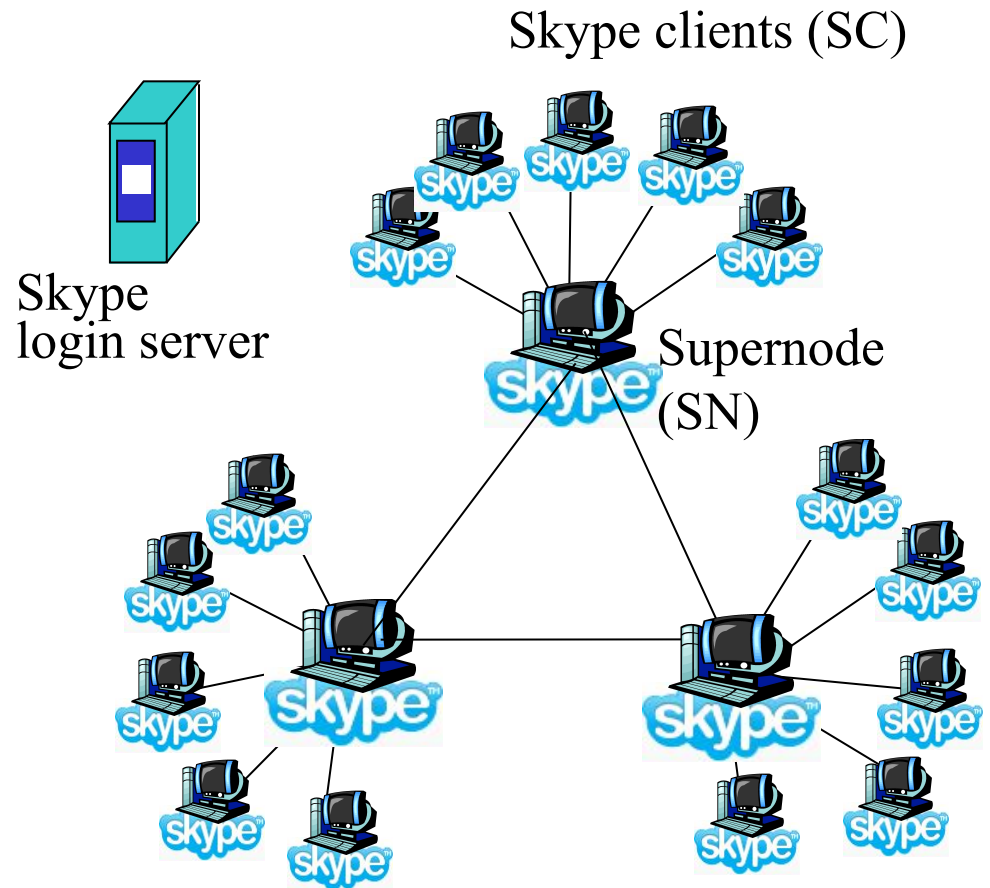
Distribuição de arquivo P2P: BitTorrent

- r par que se une à torrente:
 - m não tem nenhum bloco, mas irá acumulá-los com o tempo
 - m registra com o *tracker* para obter lista dos pares, conecta a um subconjunto de pares ("vizinhos")
- r enquanto faz o download, par carrega blocos para outros pares
- r par pode mudar os parceiros com os quais troca os blocos
- r pares podem entrar e sair
- r quando o par obtiver todo o arquivo, ele pode (egoisticamente) sair ou permanecer (altruisticamente) na torrente



Estudo de caso P2P: Skype

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



Capítulo 2: Roteiro

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

Streaming de vídeo e CDNs: contexto

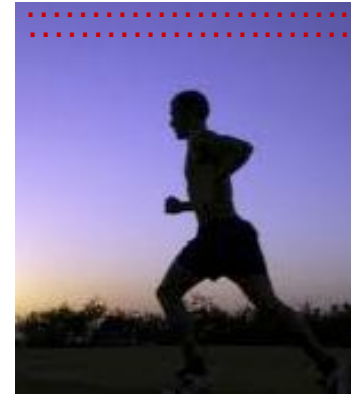
- Tráfego de vídeo: maior consumidor de largura de banda da Internet
 - Netflix, YouTube: 37%, 16% do tráfego *downstream* residencial dos ISPs
 - ~1B usuários do YouTube, ~75M usuários do Netflix
- desafio: escala – como alcançar ~1B usuários?
 - um único mega-vídeo server não daria conta (por quê?)
- 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

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

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



quadro $i+1$

Redes de Distribuição de Conteúdo (CDNs)

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

r opção 1: grande “mega-servidor” único

- m* ponto único de falha

- m* ponto de congestionamento de rede

- m* caminho longo distante dos clientes

- m* múltiplas cópias do vídeo enviadas pelo link de saída

....simplesmente: esta solução *não escala*

Redes de Distribuição de Conteúdo (CDNs)

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

r opção 2: armazenar/disponibilizar múltiplas cópias do vídeo em sites distribuídos geograficamente (**CDN**)

m ir fundo: colocar servidores CDN em muitas redes de acesso

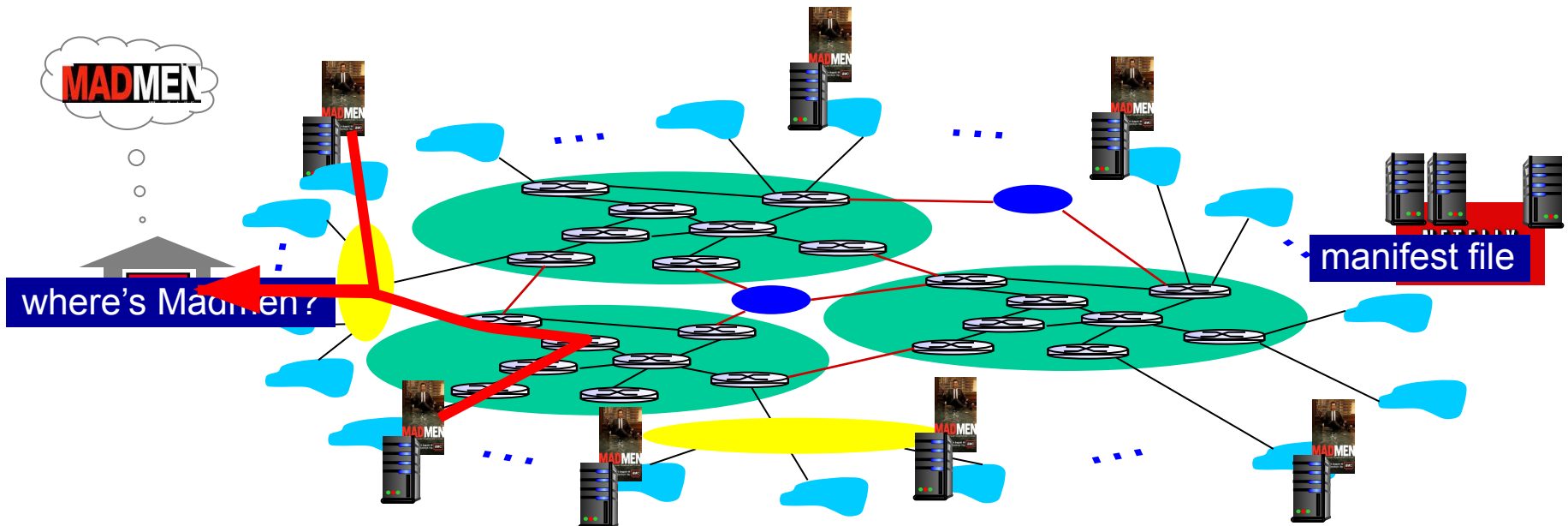
- próximo aos usuários
- usado pela Akamai, 1700 localidades

m 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)

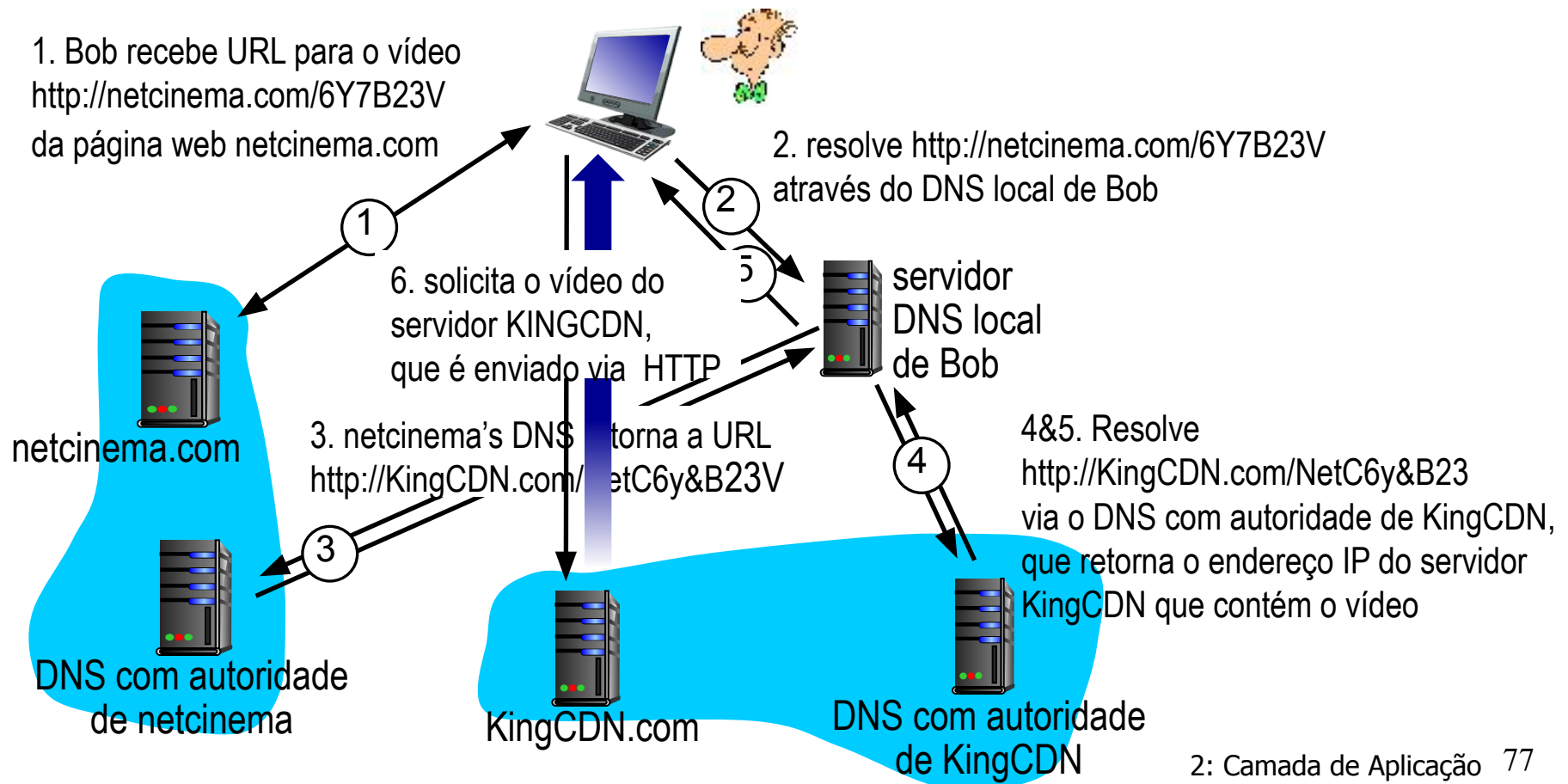
- CDN: armazena cópias do conteúdo em nós
 - e.g. Netflix armazena cópias de MadMen
- assinante solicita conteúdo da CDN
 - é direcionado para cópia mais próxima, recupera o conteúdo
 - pode escolher outra cópia, se o caminho estiver congestionado



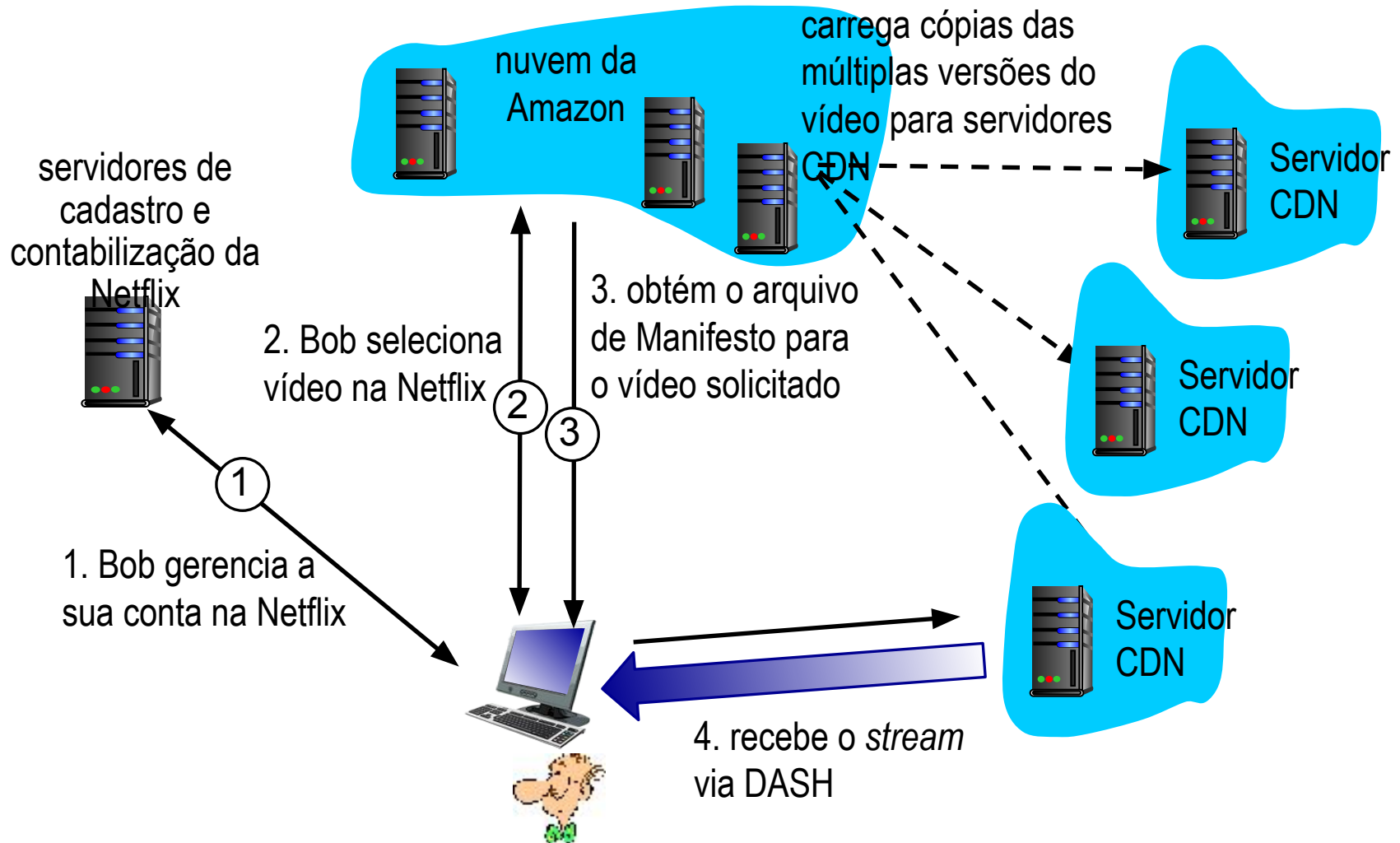
acesso a conteúdo CDN: detalhes

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

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



Caso de estudo: Netflix



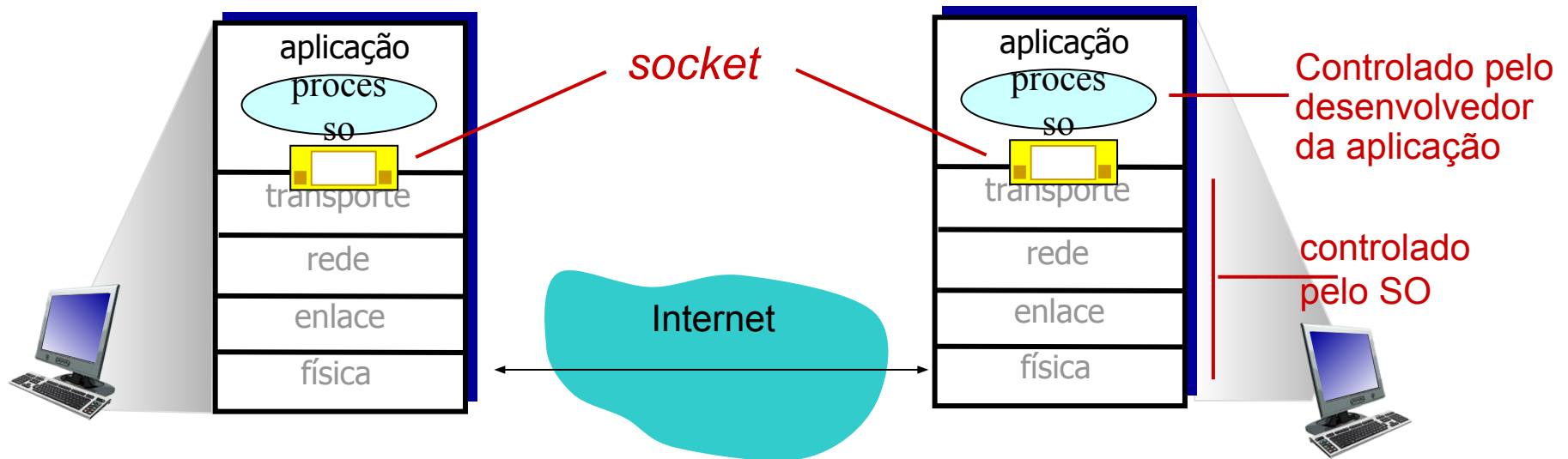
Capítulo 2: Roteiro

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

- r* **UDP:** datagrama não confiável
- r* **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

- r não tem saudação ("handshaking") antes de enviar os dados
- r remetente coloca explicitamente endereço IP e porta do destino
- r 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

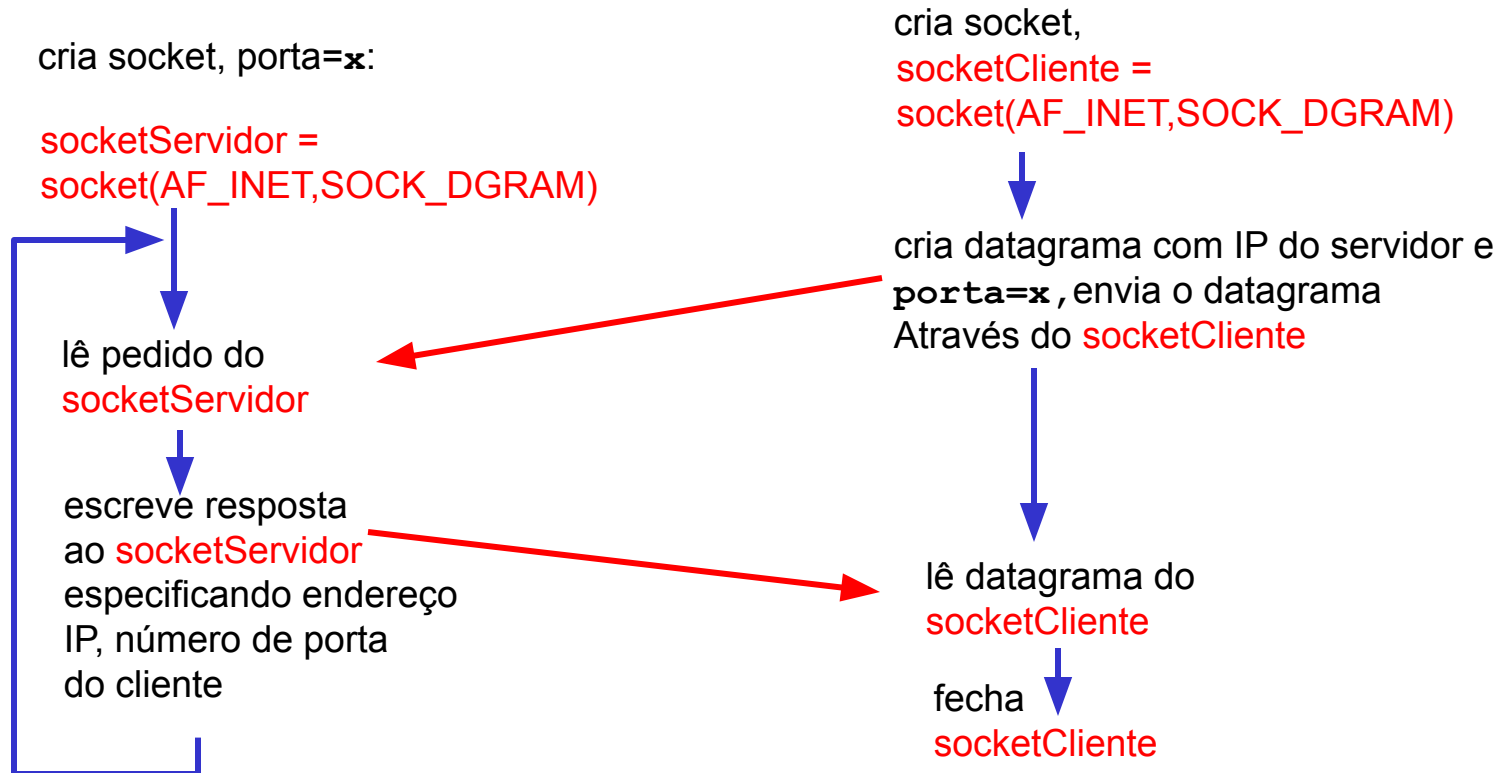
Ponto de vista da aplicação

- r 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

from socket import *

serverName = 'hostname'

serverPort = 12000

cria socket UDP para
servidor

clientSocket = socket(socket.AF_INET,
socket.SOCK_DGRAM)

obtém entrada do teclado do
usuário

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

message = raw_input('Input lowercase sentence:')

clientSocket.sendto(message,(serverName, serverPort))

lê caracteres de resposta
do socket e converte em
string

modifiedMessage, serverAddress =

imprime *string* recebido e
fecha socket

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

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

Cliente contacta servidor para:

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

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

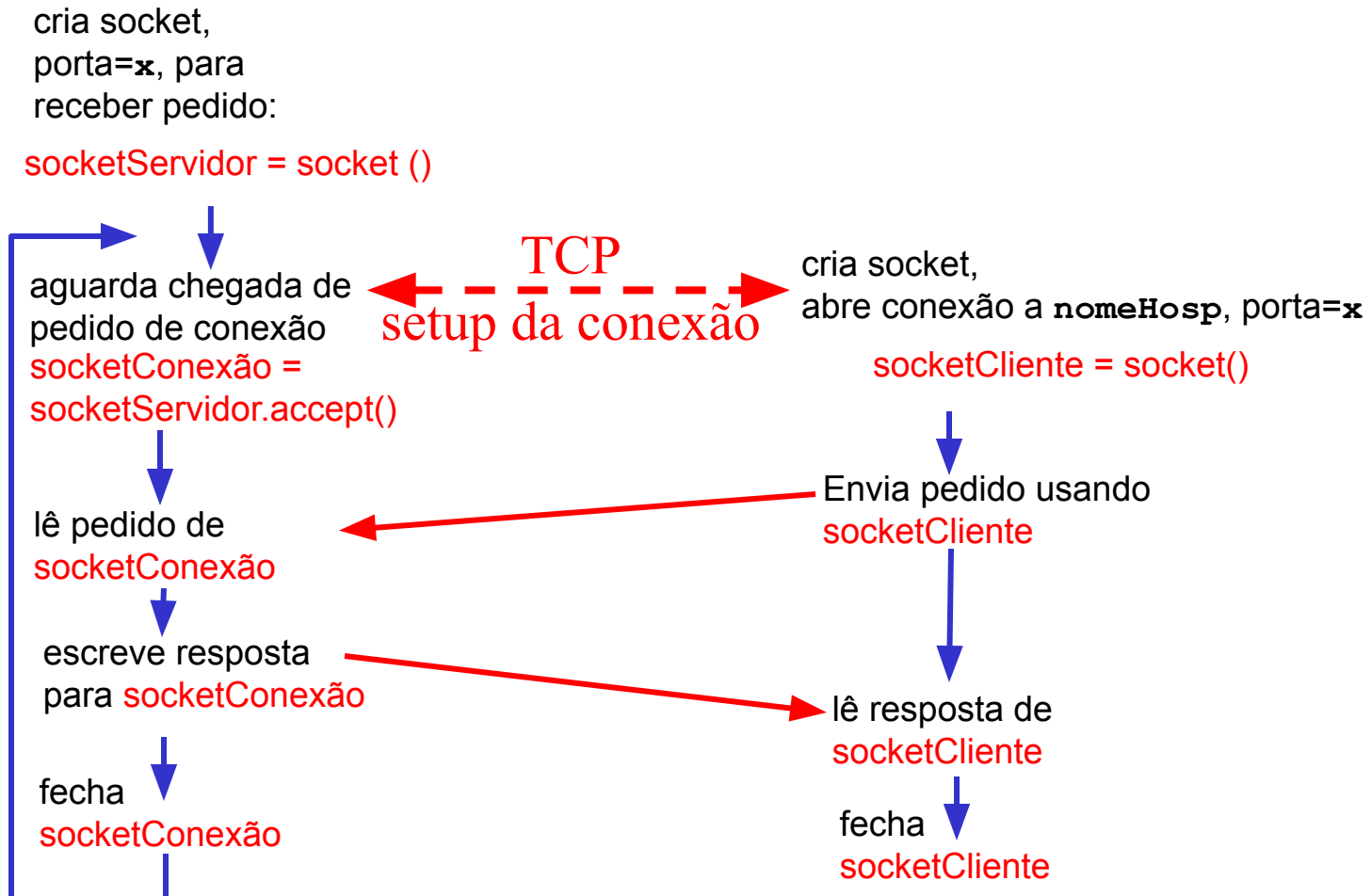
ponto de vista da aplicação

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

Interações cliente/servidor usando o TCP

Servidor (executando em `nomeHosp`)

Cliente



Exemplo: cliente TCP

inclui a biblioteca de sockets
do Python

→ `from socket import *`

`serverName = 'servername'`

`serverPort = 12000`

cria socket TCP socket
para o servidor, porta
remota 12000

→ `clientSocket = socket(AF_INET, SOCK_STREAM)`

`clientSocket.connect((serverName,serverPort))`

`sentence = raw_input('Input lowercase sentence:')`

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

→ `clientSocket.send(sentence.encode())`

`modifiedSentence = clientSocket.recv(1024)`

`print ('From Server:', modifiedSentence.decode())`

`clientSocket.close()`

Exemplo: servidor TCP

	from socket import *
cria socket TCP de recepção	serverPort = 12000
	serverSocket = socket(AF_INET, SOCK_STREAM)
servidor inicia a escuta por solicitações TCP	serverSocket.bind(('', serverPort))
	serverSocket.listen(1)
	print 'The server is ready to receive'
loop infinito	while True:
servidor espera no accept() por solicitações, um novo socket é criado no retorno	connectionSocket, addr = serverSocket.accept()
lê bytes do socket (mas não precisa ler endereço como no UDP)	sentence = connectionSocket.recv(1024).decode() capitalizedSentence = sentence.upper() connectionSocket.send(capitalizedSentence. encode())
fecha conexão para este cliente (mas <i>não</i> o socket de recepção)	connectionSocket.close()

Capítulo 2: Resumo

Nosso estudo sobre aplicações de rede está agora completo!

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

Capítulo 2: Resumo

Mais importante: aprendemos sobre protocolos

r troca típica de mensagens pedido/resposta

- m cliente solicita info ou serviço
- m servidor responde com dados, código de *status*

r formatos de mensagens:

- m **cabeçalhos**: campos com info sobre dados (metadados)
- m **dados**: info (carga) sendo comunicada

Temas importantes:

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