

Redes de Computadores

Camada de Aplicação

Nota: a maioria dos slides dessa apresentação são traduzidos ou adaptados dos slides disponibilizados gratuitamente pelos autores do livro [KUROSE, James F. e ROSS, Keith W. *Computer Networking: A Top-Down Approach.* 8th Edition Pearson, 2020.](#) Todo o material pertencente aos seus respectivos autores está protegido por direito autoral.

Camada de aplicação: visão geral

- princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- o Domain Name System DNS
- aplicações P2P
- transmissão de vídeo e redes de distribuição de conteúdo



Camada de aplicação: visão geral

Nossos objetivos:

- compreender aspectos conceituais e de implementação de protocolos da camada de aplicação
 - modelos de serviço de camada de transporte
 - paradigma cliente-servidor
 - paradigma ponto a ponto

- aprender sobre protocolos examinando protocolos populares da camada de aplicação e sua infraestrutura
 - HTTP
 - SMTP, IMAP
 - DNS
 - sistemas de streaming de vídeo, CDNs
- programação de aplicativos de rede
 - API de sockets

Algumas aplicações de rede

- redes sociais
- Web
- mensagens de texto
- e-mail
- jogos de rede multiusuário
- streaming de vídeo armazenado (YouTube, Hulu, Netflix)
- compartilhamento de arquivos P2P
- voz sobre IP (ex.: Skype)
- videoconferência em tempo real (ex: Zoom)
- busca na Internet
- login remoto
- ...

Q: suas favoritas?

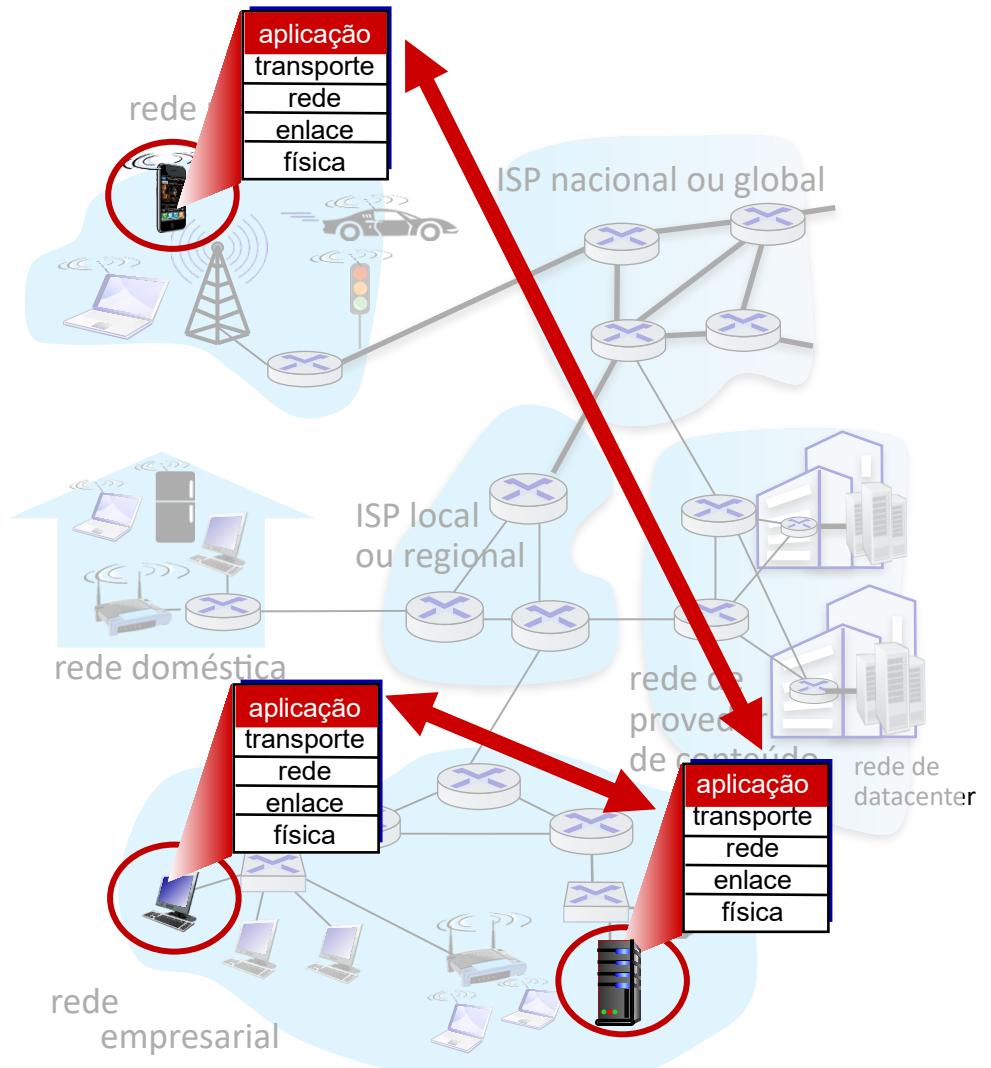
Criação de um aplicativo de rede

escreva programas que:

- executem em sistemas finais (diferentes)
- se comuniquem pela rede
- ex.: software de servidor web se comunica com software de navegador web

não há necessidade de escrever software para dispositivos do núcleo da rede

- dispositivos do núcleo da rede não executam aplicativos de usuário
- aplicativos executando em sistemas finais permitem o desenvolvimento e propagação rápidos de novas aplicações



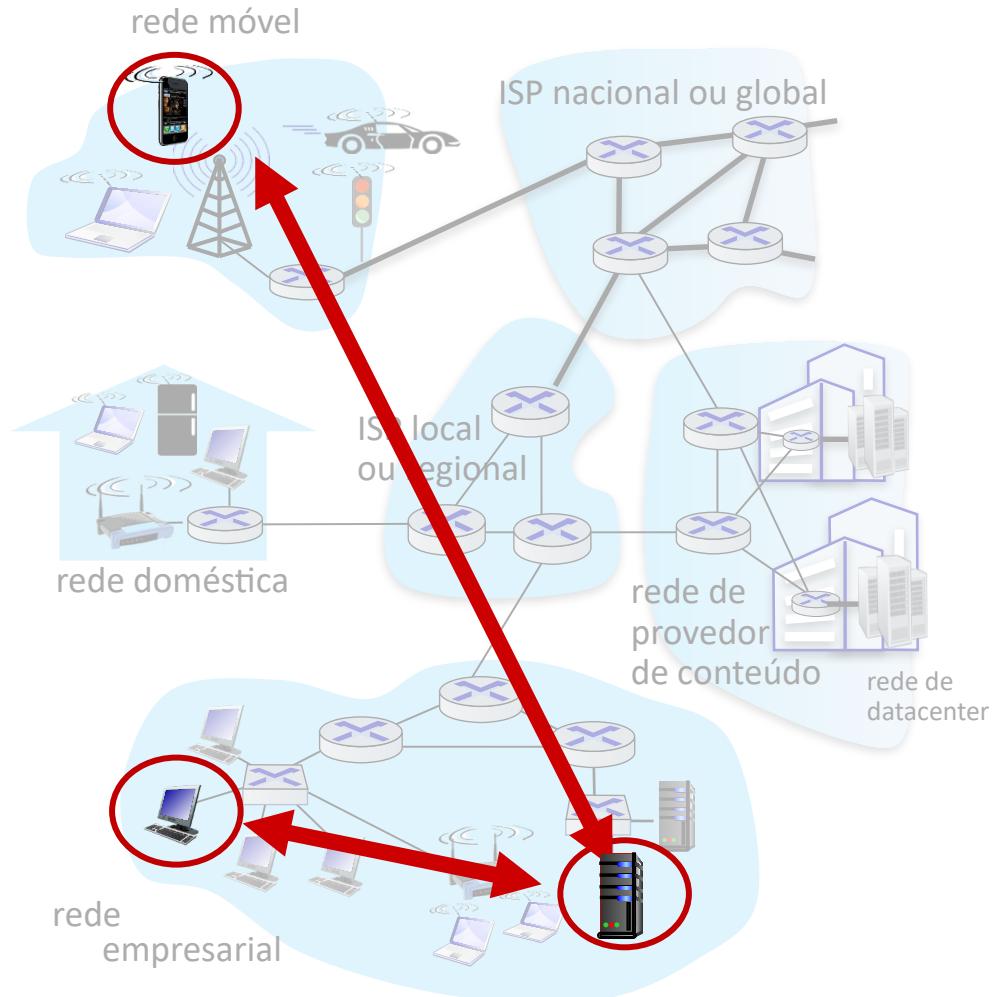
Paradigma cliente-servidor

servidor:

- hospedeiro sempre ligado
- endereço IP permanente
- frequentemente em *data centers*, para escalar

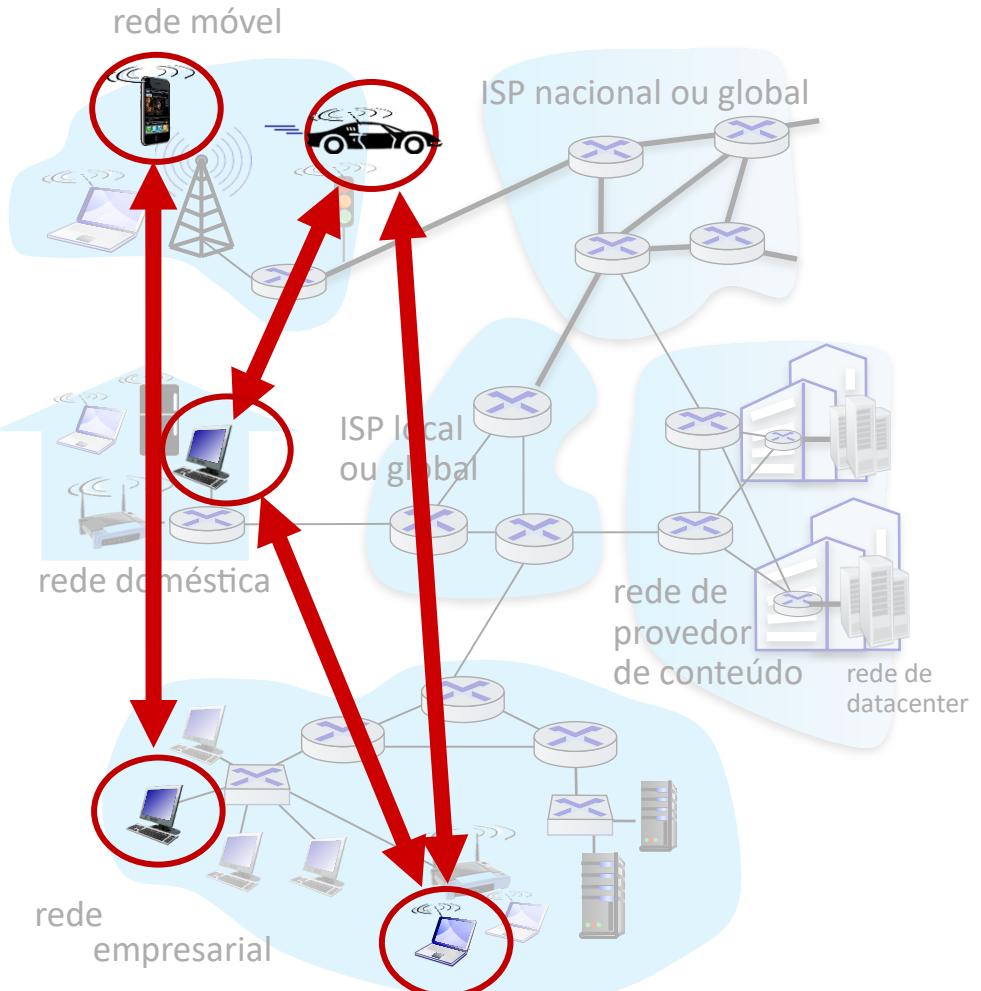
clientes:

- contatam e se comunicam com o servidor
- pode estar conectado de forma intermitente
- pode ter endereços IP dinâmicos
- *não* se comunicam diretamente uns com os outros
- exemplos: HTTP, IMAP, FTP



Arquitetura peer-to-peer

- *nenhum* servidor sempre ligado
- sistemas finais arbitrários se comunicam diretamente
- pares solicitam serviço de outros pares, e em troca fornecem serviço para outros pares
 - *auto escalabilidade* – novos pares trazem nova capacidade de serviço, bem como novas demandas de serviço
- os pares estão conectados de forma intermitente e mudam os endereços IPs
 - gerenciamento complexo
- exemplo: compartilhamento de arquivos P2P



Comunicação entre processos

processo: programa executando em um hospedeiro

- dentro do mesmo hospedeiro, dois processos se comunicam usando **comunicação entre processos** (definido pelo SO)
- processos em diferentes hospedeiros se comunicam trocando **mensagens**

clientes, servidores

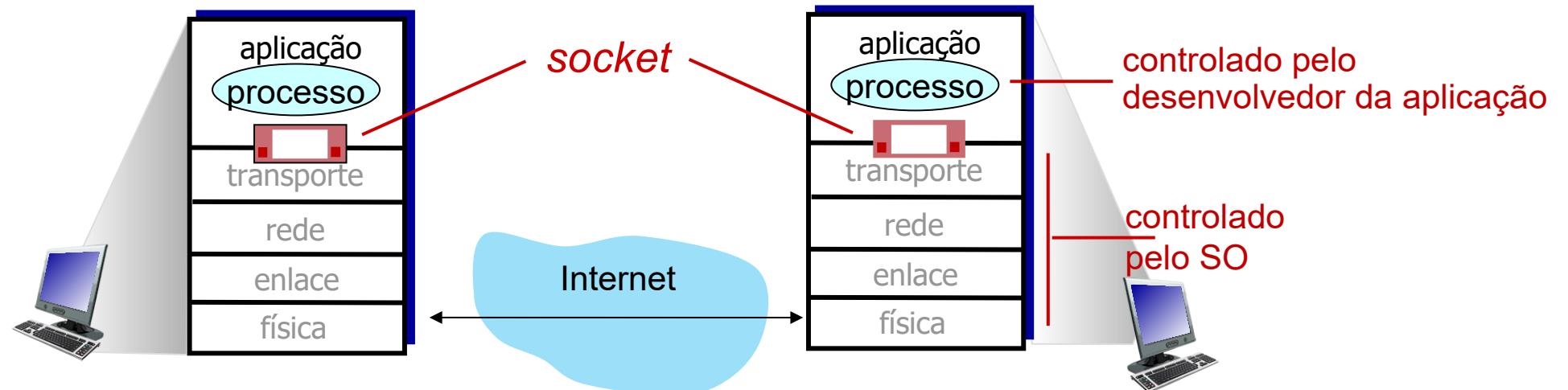
processo cliente: processo que inicia a comunicação

processo servidor: processo que espera ser contatado

- nota: aplicações com arquitetura P2P tem processos clientes e processos servidores

Sockets

- processo envia/recebe mensagens para/de **socket**
- socket análogo à porta
 - processo remetente empurra mensagem porta afora
 - processo remetente depende da infraestrutura de transporte do outro lado da porta para entregar a mensagem ao socket no processo destinatário
 - dois sockets envolvidos: um de cada lado



Endereçando processos

- para receber mensagens, processo deve ter um *identificador*
- dispositivo hospedeiro tem endereço IP exclusivo de 32 bits
- *Q:* o endereço IP do hospedeiro em que o processo é executado é suficiente para identificar o processo?
 - *R:* não, *muitos* processos podem ser executados no mesmo hospedeiro
- *identificador* inclui ambos **endereços IP** e **números de porta** associados com o processo no hospedeiro.
- exemplos de números de porta:
 - servidor HTTP: 80
 - servidor de e-mail: 25
- para enviar uma mensagem HTTP para o servidor web `gaia.cs.umass.edu`:
 - **endereço IP:** 128.119.245.12
 - **número de porta:** 80
- mais em breve...

Um protocolo de camada de aplicação define:

- tipos de mensagens trocadas,
 - ex.: requisição, resposta
- sintaxe da mensagem :
 - quais campos nas mensagens e como os campos são delineados
- semântica da mensagem
 - significado da informação nos campos
- regras para saber quando e como os processos enviam e respondem às mensagens

protocolos abertos:

- definido em RFCs, todos têm acesso às definições do protocolo
- permite interoperabilidade
- ex.: HTTP, SMTP

protocolos proprietários:

- ex.: Skype, Zoom

Qual serviço de transporte um aplicativo precisa?

integridade de dados

- algumas aplicações (por exemplo, transferência de arquivos, transações da web) exigem transferência de dados 100% confiável
- outras aplicações (por exemplo, áudio) podem tolerar alguma perda

temporização

- algumas aplicações (por exemplo, telefonia pela Internet, jogos interativos) requerem baixa latência para serem “eficazes”

vazão

- algumas aplicações (por exemplo, multimídia) exigem uma quantidade mínima de vazão para serem “eficazes”
- outras aplicações (“aplicações elásticas”) fazem uso de qualquer vazão que obtêm

segurança

- criptografia, integridade de dados, ...

Requisitos do serviço de transporte: aplicações comuns

aplicação	perda de dados	vazão	sensibilidade ao tempo
transferência de arquivos	sem perda	elástico	não
e-mail	sem perda	elástico	não
documentos Web	sem perda	elástico	não
áudio/vídeo em tempo real	tolerante a perda	áudio: 5kbps-1Mbps vídeo:10kbps-5Mbps	sim, dezenas de milissegundos
áudio/vídeo armazenado	tolerante a perda	mesmo que acima	sim, alguns segundos
jogos interativos	tolerante a perda	poucos kbps ou mais	sim, dezenas de milissegundos
mensagens instantâneas	sem perda	elástico	sim e não

Serviços dos Protocolos de Transporte da Internet

serviço TCP:

- *transporte confiável* entre processos emissor e receptor
- *controle de fluxo*: emissor não vai sobrecarregar o receptor
- *controle de congestionamento*: desacelera o emissor quando a rede está sobrecarregada
- *orientado a conexão*: configuração necessária entre os processos cliente e servidor
- *não oferece*: temporização, garantia de vazão mínima, segurança

serviço UDP:

- *transferência de dados não confiável* entre processos emissor e receptor
- *não oferece*: confiabilidade, controle de fluxo, controle de congestionamento, temporização, garantia de vazão, ou configuração de conexão.

Q: porque se importar? Por que existe um UDP?

Aplicações de Internet e protocolos de transporte

aplicação	protocolo da camada de aplicação	protocolo de transporte
transferência de arquivos / download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
documentos Web	HTTP 1.1 [RFC 7320]	TCP
telefonia na Internet	SIP [RFC 3261], RTP [RFC 3550], ou proprietário	TCP ou UDP
streaming de áudio/vídeo	HTTP [RFC 7320], DASH	TCP
jogos interativos	WOW, FPS [proprietário]	UDP ou TCP

Deixando o TCP Seguro

Sockets TCP & UDP tradicionais:

- sem criptografia
- as senhas de texto não criptografado enviadas para o soquete atravessam a Internet em texto não criptografado (!)

Transport Layer Security (TLS)

- fornece conexões TCP criptografadas
- integridade de dados
- autenticação ponto-a-ponto

TLS implementado na camada de aplicação

- aplicações usam bibliotecas TLS, que por sua vez usam TCP
- Texto não criptografado enviado para o “socket” atravessa a Internet *criptografado*
- mais: Capítulo 8

Camada de aplicação: visão geral

- princípios de aplicações de rede
- **Web e HTTP**
- E-mail, SMTP, IMAP
- o Domain Name System
DNS
- aplicações P2P
- transmissão de vídeo e redes de distribuição de conteúdo



Web e HTTP

Primeiro, uma revisão rápida...

- página web consiste de *objetos*, cada um deles pode ser armazenado em servidores Web diferentes
- objetos podem ser arquivos HTML, imagens JPEG, applets Java (!!), arquivos de áudio,...
- página web consiste de *arquivo HTML base* o qual inclui *vários objetos referenciados, cada um* endereçável por uma *URL*, ex.:

www.someschool.edu/someDept/pic.gif

nome do hospedeiro

nome do caminho

Visão geral do HTTP

HTTP: hypertext transfer protocol

- protocolo da camada de aplicação da Web
- modelo cliente/servidor:
 - *cliente*: navegador que solicita, recebe (usando o protocolo HTTP) e “exibe” objetos da Web
 - *servidor*: o servidor Web envia (usando o protocolo HTTP) objetos em resposta a solicitações



Visão geral do HTTP (continuação)

HTTP usa TCP:

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

HTTP é “sem estado”

- o servidor *não* mantém nenhuma informação sobre as solicitações anteriores do cliente

nota

protocolos que mantém “estado” são complexos!

- história passada (estado) deve ser mantida
- se o servidor/cliente travar, suas visões de “estado” podem ser inconsistentes, devem ser reconciliadas

Conexões HTTP: dois tipos

HTTP não persistente

1. conexão TCP aberta
2. no máximo um objeto enviado pela conexão TCP
3. conexão TCP fechada
baixar vários objetos requer várias conexões

HTTP persistente

- conexão TCP aberta com um servidor
- vários objetos podem ser enviados por meio de uma *única* conexão TCP entre o cliente e esse servidor
- conexão TCP fechada

HTTP não persistente: exemplo

Usuário digita URL: `www.someSchool.edu/someDepartment/home.index`
(contendo texto, referências a 10 imagens jpeg)



- tempo ↓
- 1a. cliente HTTP inicia conexão TCP para o servidor (processo) em `www.someSchool.edu` na porta 80
 - 1b. servidor HTTP no hospedeiro `www.someSchool.edu` esperando por conexões TCP na porta 80 “aceita” conexão, notificando cliente
 2. cliente HTTP envia *mensagem de requisição* HTTP (contendo URL) no socket da conexão TCP. Mensagem indica que o cliente quer o objeto `someDepartment/home.index`
 3. servidor HTTP recebe mensagem de requisição, forma *mensagem de resposta* contendo o objeto requisitado, e envia a mensagem em seu soquete

HTTP não persistente: exemplo (cont.)

Usuário digita URL: `www.someSchool.edu/someDepartment/home.index`
(contendo texto, referências a 10 imagens jpeg)



tempo ↓

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

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

4. o servidor HTTP fecha a conexão TCP.

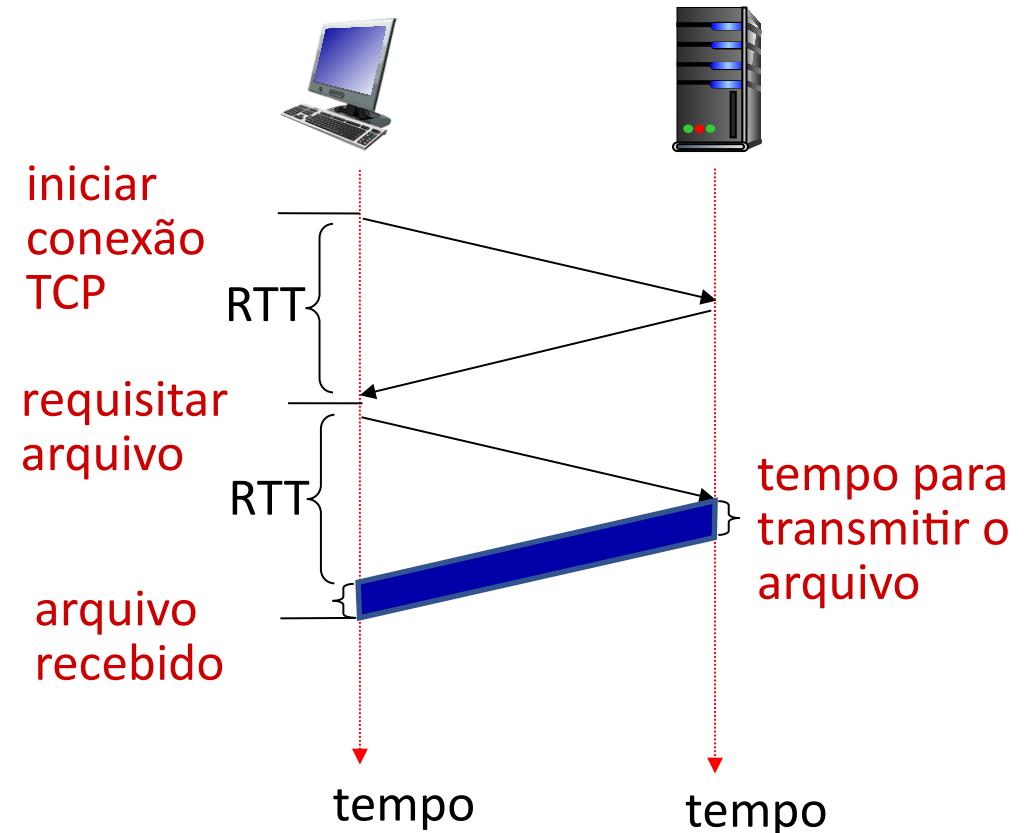


HTTP não persistente: tempo de resposta

RTT (definição): tempo para um pequeno pacote viajar do cliente para o servidor e voltar

tempo de resposta do HTTP (por objeto):

- um RTT para iniciar a conexão TCP
- um RTT para enviar a requisição HTTP e receber os primeiros bytes da resposta HTTP
- tempo de transmissão do objeto/arquivo



$$\text{Tempo de resposta do HTTP não persistente} = 2\text{RTT} + \text{tempo de transmissão do arquivo}$$

HTTP Persistente (HTTP 1.1)

limitações do HTTP não persistente:

- requer 2 RTTs por objeto
- sobrecarga do SO para *cada* conexão TCP
- navegadores costumam abrir várias conexões TCP paralelas para buscar objetos referenciados em paralelo

HTTP persistente (HTTP1.1):

- o servidor deixa a conexão aberta após enviar a resposta
- mensagens HTTP subsequentes entre o mesmo cliente/servidor são enviadas por meio da conexão aberta
- o cliente envia solicitações assim que encontra um objeto referenciado
- pode haver apenas um RTT para todos os objetos referenciados (reduzindo o tempo de resposta pela metade)

Mensagem de requisição HTTP

- dois tipos de mensagens HTTP: *requisição, resposta*
- mensagem de requisição HTTP:**

- ASCII (formato legível por humanos)

linha de
requisição
(commandos
GET, POST,
HEAD)

linhas de
cabeçalho

retorno de carro, avanço
de linha no início da
linha indica o fim das
linhas do cabeçalho

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \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
Connection: keep-alive\r\n
\r\n
```

caractere de retorno de carro

caractere de alimentação de linha

Outras mensagens de solicitação HTTP

método POST:

- página da web frequentemente inclui formulário de entrada
- entrada do usuário enviada do cliente para o servidor no corpo da entidade da mensagem de requisição HTTP POST

método GET (para enviar dados para o servidor):

- inclui dados do usuário no campo URL da mensagem de requisição HTTP GET (seguindo um '?'):

www.somesite.com/animalsearch?monkeys&banana

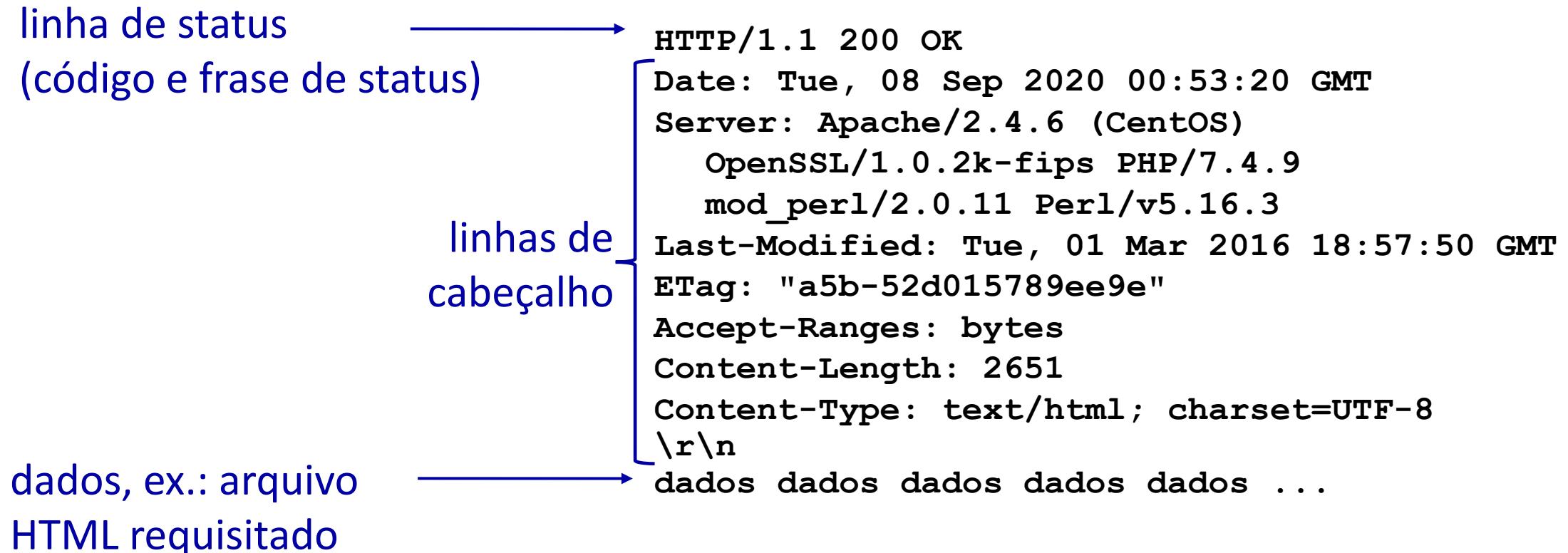
método HEAD:

- solicita (apenas) cabeçalhos que seriam retornados se a URL especificada fosse solicitada com o método HTTP GET.

método PUT:

- envia um novo arquivo (objeto) para o servidor
- substitui completamente o arquivo que existe na URL especificada pelo conteúdo no corpo da entidade da mensagem de requisição HTTP POST

Mensagem de resposta HTTP



* Confira os exercícios interativos online para mais exemplos: http://gaia.cs.umass.edu/kurose_ross/interactive/

Códigos de status de resposta HTTP

- o código de status aparece na primeira linha na mensagem de resposta do servidor para o cliente.
- alguns códigos de exemplo:

200 OK

- solicitação bem-sucedida, objeto solicitado adiante nesta mensagem

301 Moved Permanently

- objeto solicitado movido, novo local especificado adiante nesta mensagem (no campo Local:)

400 Bad Request

- mensagem de requisição não compreendida pelo servidor

404 Not Found

- documento solicitado não encontrado neste servidor

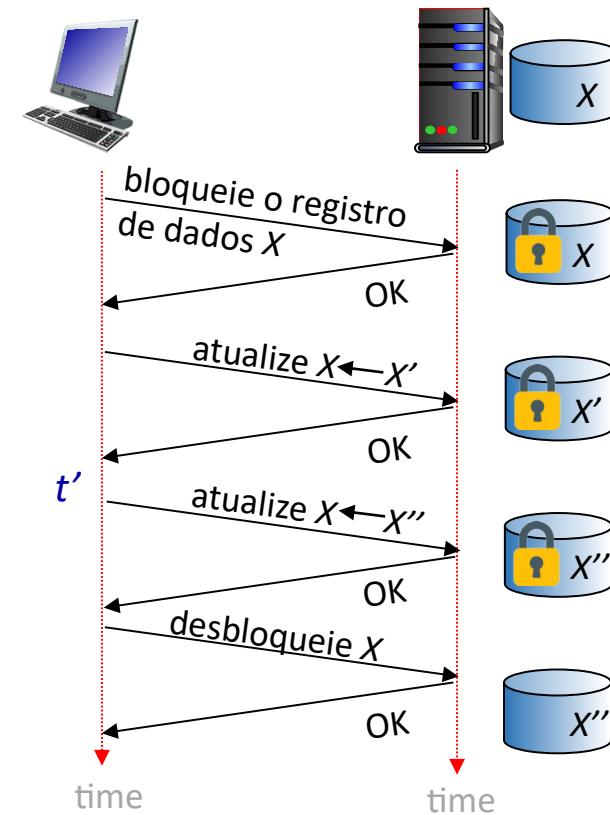
505 HTTP Version Not Supported

Mantendo o estado do usuário/servidor: cookies

Lembre-se: a interação HTTP
GET/resposta *não tem estado*

- nenhuma noção de troca em várias etapas de mensagens HTTP para concluir uma “transação da Web”
 - não há necessidade do cliente/servidor rastrear o “estado” da troca de várias etapas
 - todas as solicitações HTTP são independentes umas das outras
 - não há necessidade de o cliente/servidor “se recuperar” de uma transação parcialmente concluída, mas nunca completamente concluída

um protocolo com estado: cliente faz duas mudanças em X, ou nenhuma



Q: o que acontece se a conexão de rede ou o cliente travar em t' ?

Mantendo o estado do usuário/servidor: cookies

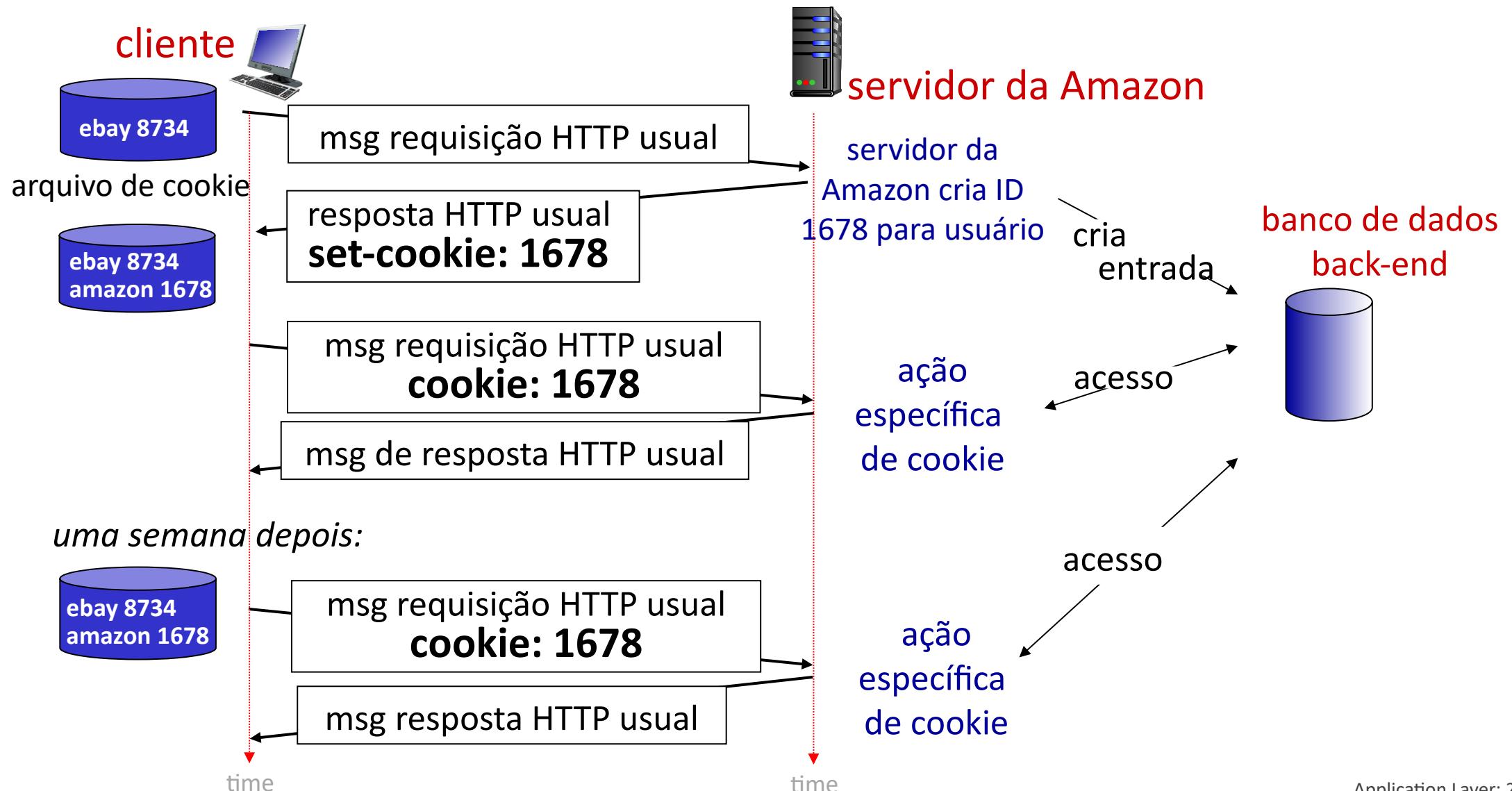
Sites e navegador do cliente usam *cookies* para manter algum estado entre as transações *quatro componentes*:

- 1) linha de cabeçalho do cookie na mensagem de *resposta* HTTP
- 2) linha de cabeçalho do cookie na próxima mensagem de *solicitação* HTTP
- 3) arquivo de cookie mantido no hospedeiro do usuário, gerenciado pelo navegador do usuário
- 4) banco de dados *back-end* no site Web

Exemplo:

- Susan usa o navegador em um laptop e visita um site de comércio eletrônico específico pela primeira vez
- quando as solicitações HTTP iniciais chegam ao site, o site cria:
 - ID único (também conhecido como “cookie”)
 - entrada no banco de dados *back-end* para o ID
 - As solicitações HTTP subsequentes de Susan para este site conterão o valor do ID do cookie, permitindo que o site “identifique” Susan

Mantendo o estado do usuário/servidor: cookies



Cookies HTTP: comentários

Para que cookies podem ser usados:

- autorização
- carrinhos de compras
- recomendações
- estado de sessão de usuário (Web e-mail)

Desafio: como manter o estado?

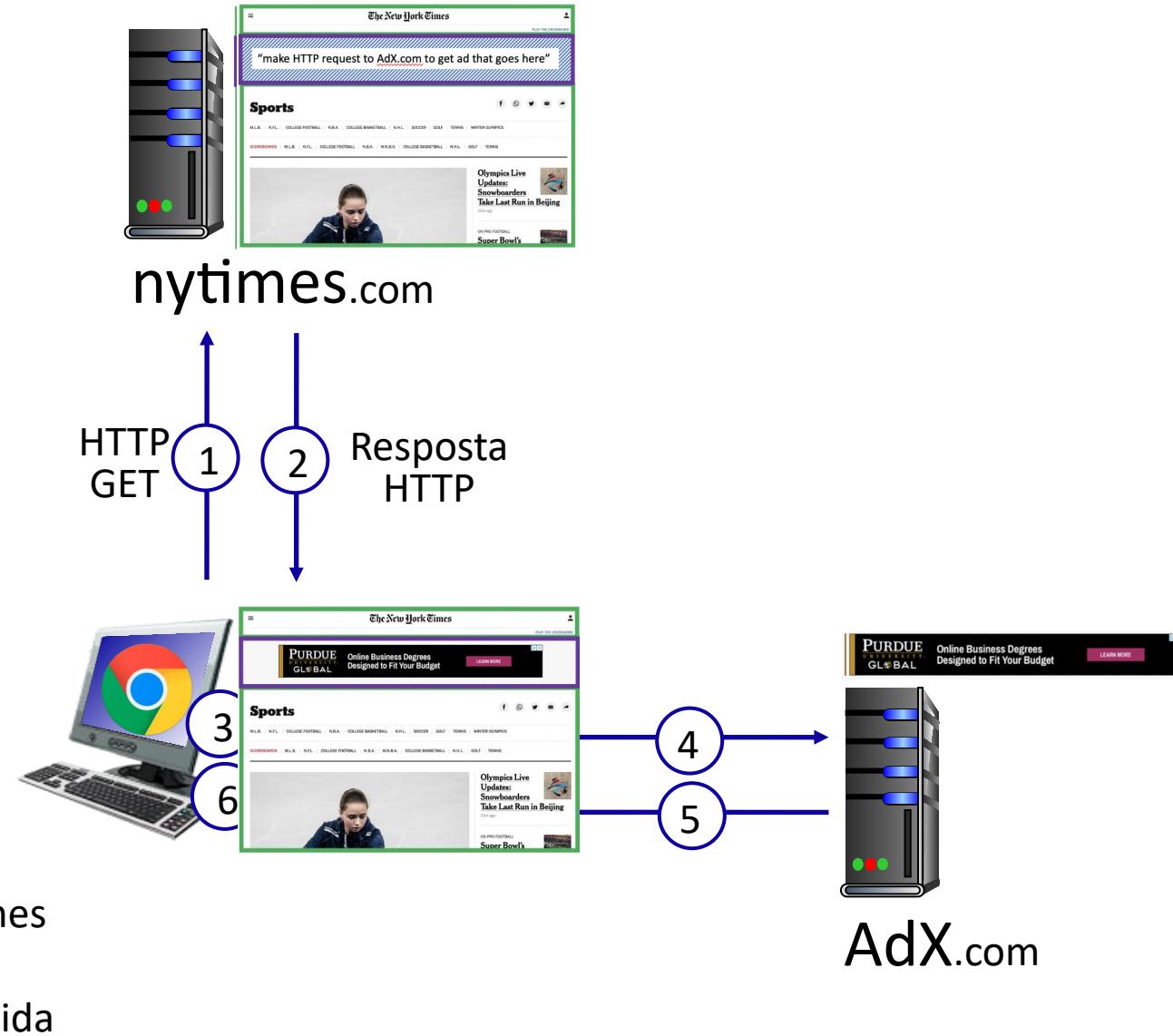
- *nas extremidades do protocolo:*
mantém o estado no remetente/receptor por várias transações
- *em mensagens:* cookies em mensagens HTTP carregam o estado

nota
cookies e privacidade:

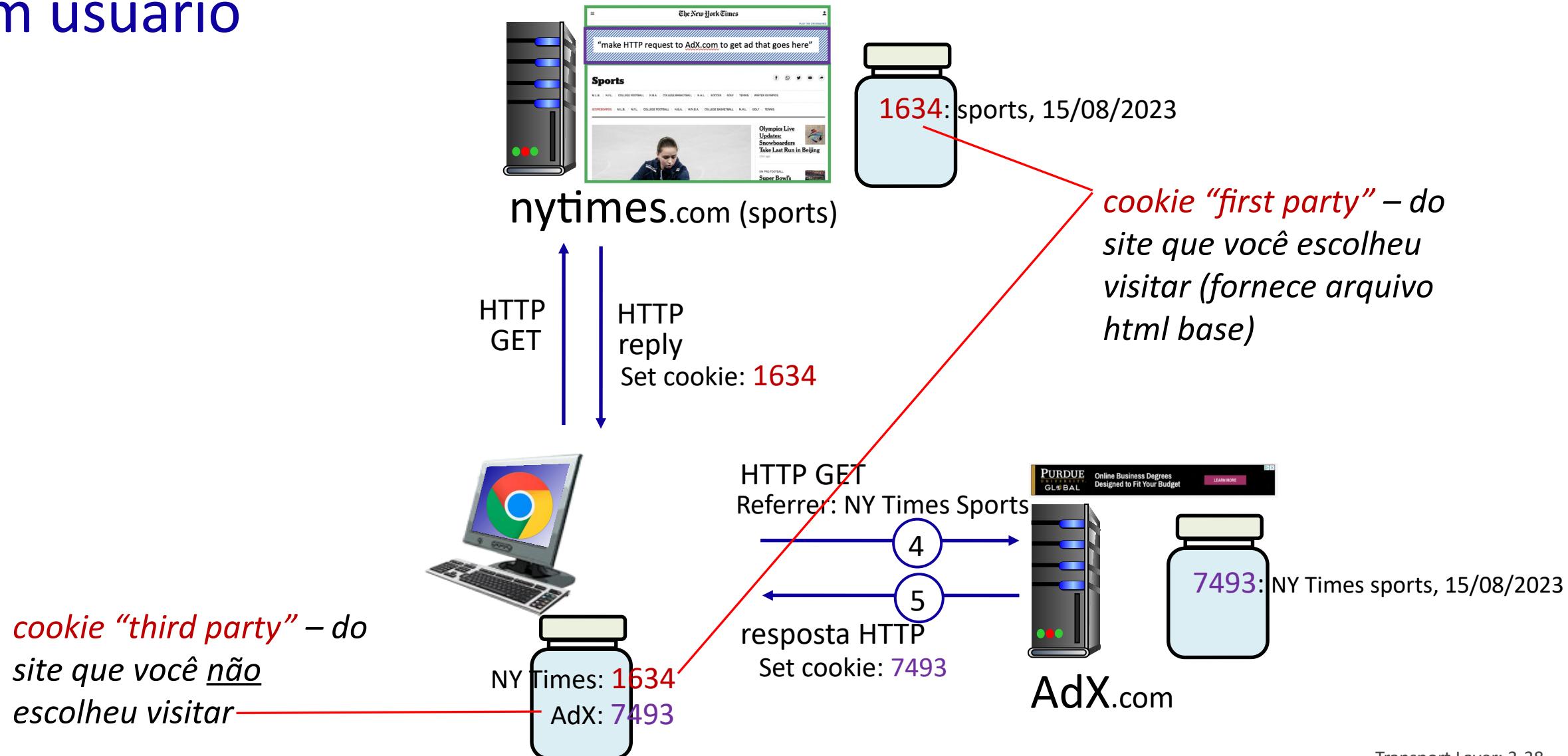
- os cookies permitem que os sites *aprendam* muito sobre você.
- cookies persistentes de terceiros (cookies de rastreamento) permitem que uma identidade comum (valor do cookie) seja rastreada em múltiplos sites

Exemplo: exibindo uma página da web do NY Times

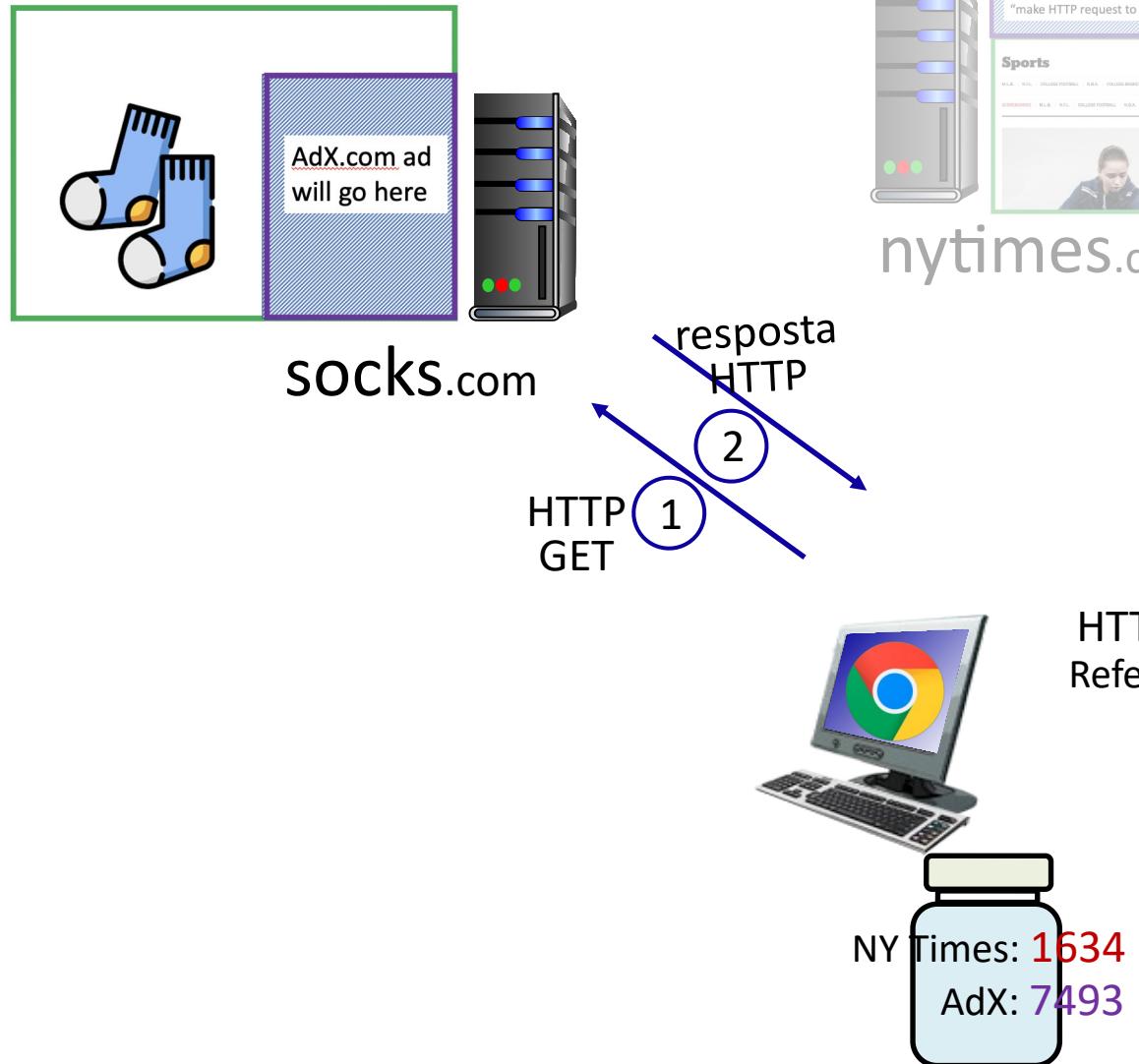
- 1 GET (obtém) arquivo html base de nytimes.com
- 2 busca anúncio de AdX.com
- 7 exibe página composta



Cookies: rastreando o comportamento de navegação de um usuário



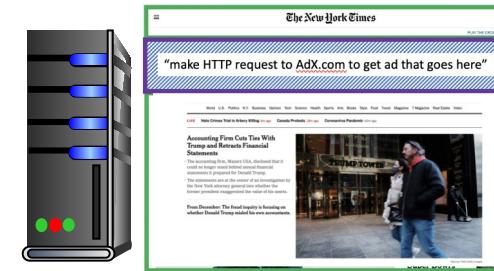
Cookies: rastreando o comportamento de navegação de um usuário



AdX:

- *rastreia minha navegação na web* em sites com anúncios do AdX
- pode retornar anúncios direcionados com base no histórico de navegação

Cookies: rastreando o comportamento de navegação de um usuário (um dia depois)



nytimes.com (arts)

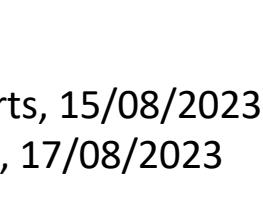
HTTP
GET
cookie: 1634

resposta
HTTP
Set cookie: 1634



HTTP GET
Referrer: NY Times Arts, cookie: 7493

4
5
HTTP reply



AdX.com



Cookies: rastreando o comportamento de navegação de um usuário

Os cookies podem ser usados para:

- rastrear o comportamento do usuário em um determinado site (**first party cookies**)
- rastrear o comportamento do usuário em vários sites (**third party cookies**) sem que o usuário escolha visitar o site do rastreador (!)
- rastreamento pode ser *invisível* para o usuário:
 - em vez de o anúncio exibido acionar o HTTP GET para o rastreador, pode ser um link invisível

rastreamento de terceiros via cookies:

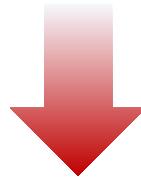
- desativado por padrão nos navegadores Firefox e Safari
- seria desativado no navegador Chrome em 2023
 - Google adiou a mudança para 2024 e depois desistiu.

GDPR (General Data Protection Regulation da Europa) e cookies

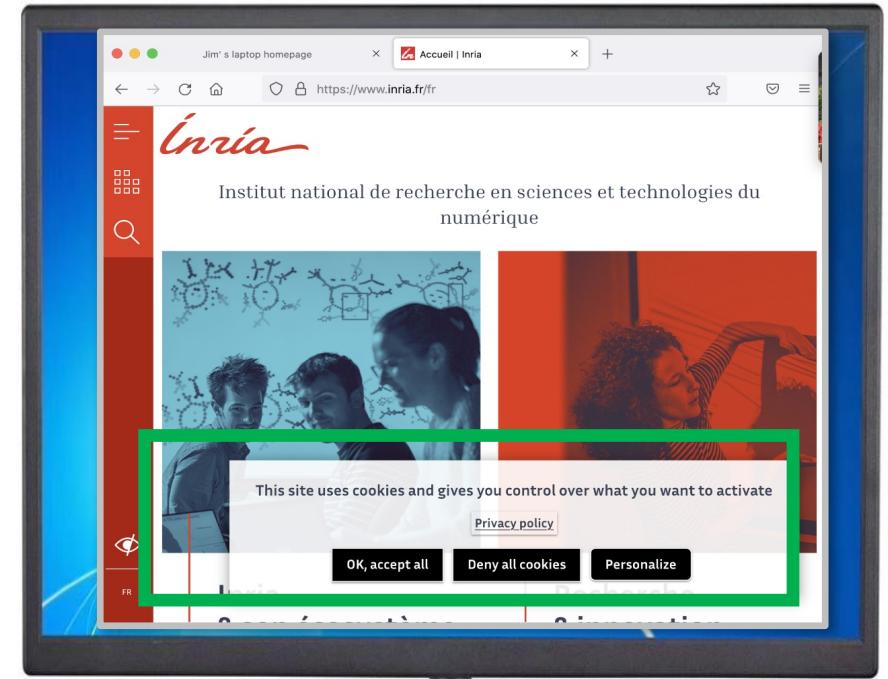
“Pessoas físicas podem estar associadas a identificadores on-line [...], como endereços de protocolo da Internet, identificadores de cookies ou outros identificadores [...].

Isso pode deixar rastros que, em particular quando combinados com identificadores únicos e outras informações recebidas pelos servidores, podem ser usados para criar perfis das pessoas físicas e identificá-las.”

GDPR, recital 30 (Maio de 2018)



quando os cookies podem identificar um indivíduo, os cookies são considerados dados pessoais, sujeitos aos regulamentos de dados pessoais do GDPR

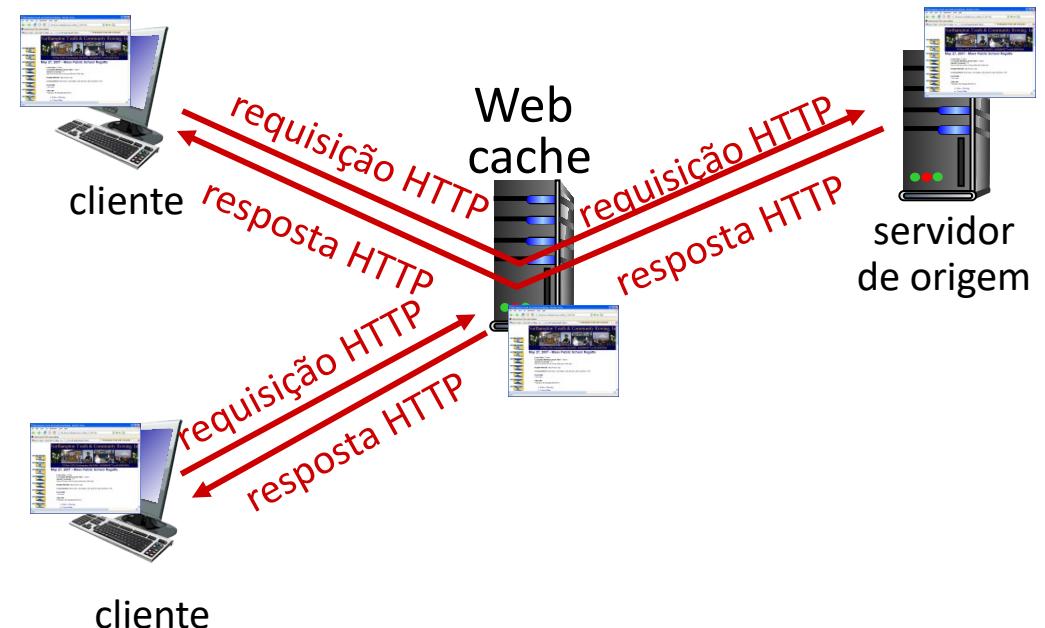


O usuário tem controle explícito sobre se os cookies são permitidos ou não

Caches Web

Objetivo: satisfazer as solicitações do cliente sem envolver o servidor de origem

- o usuário configura o navegador para apontar para um *cache Web* (local)
- navegador envia todas as solicitações HTTP para o cache
 - *se* objeto no cache: o cache retorna o objeto ao cliente
 - *senão* cache solicita o objeto do servidor de origem, armazena o objeto recebido e retorna o objeto ao cliente



Caches Web (também chamados servidores proxy)

- o cache web atua como ambos cliente e servidor
 - servidor para cliente solicitante original
 - cliente para servidor de origem
- servidor diz ao cache sobre objetos com permissão de cache no cabeçalho de resposta:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

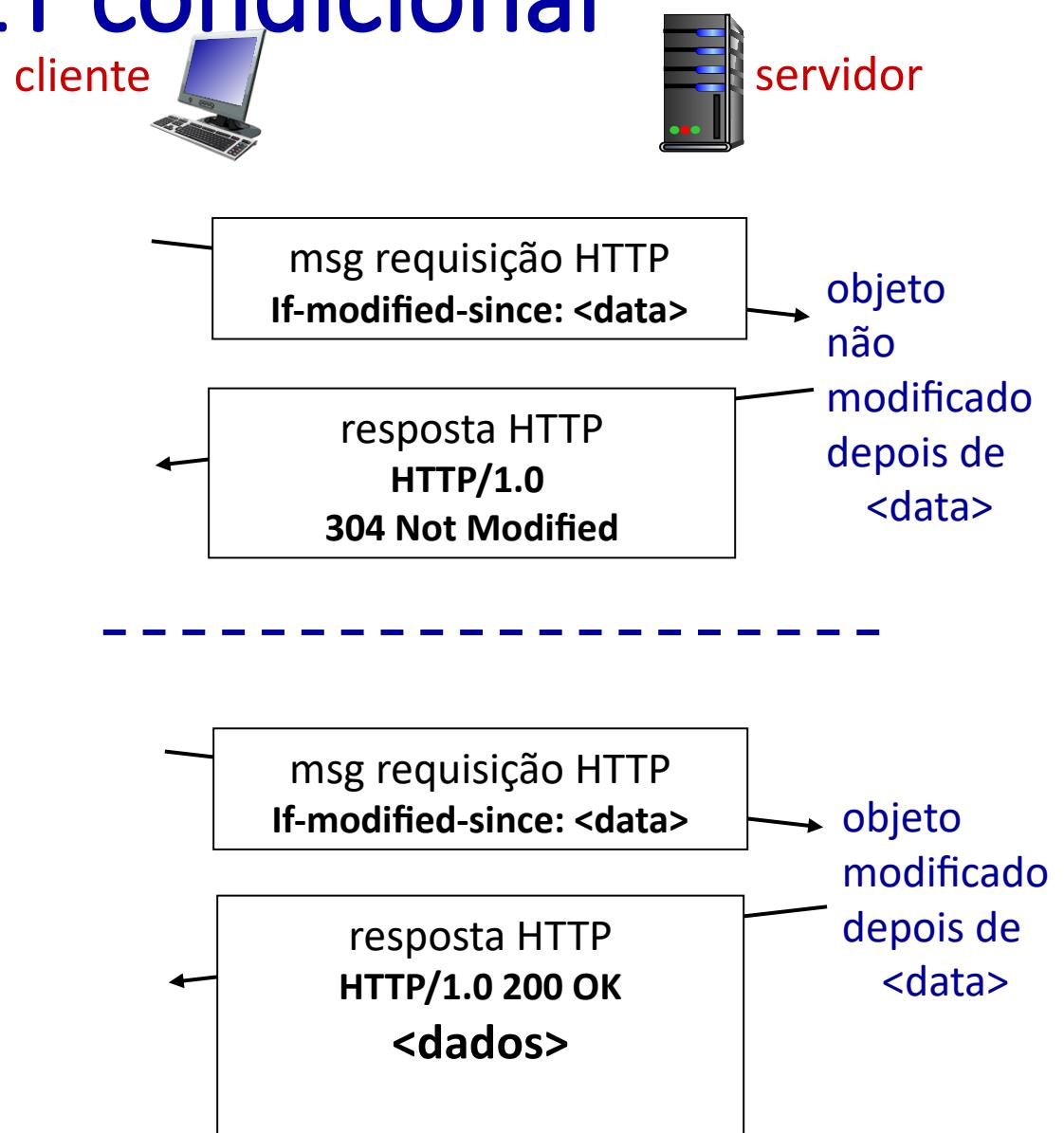
Por que cache Web?

- reduzir o tempo de resposta para solicitação do cliente
 - cache está mais perto do cliente
- reduzir o tráfego no enlace de acesso de uma instituição
- Internet é densa com caches
 - permite que provedores de conteúdo “pobres” ofereçam conteúdo de forma mais eficaz

Cache do Navegador: GET condicional

Objetivo: não enviar objeto se cache do navegador tiver versão atualizada

- sem atraso na transmissão de objetos (ou uso de recursos de rede)
- *cliente*: especifica a data da cópia armazenada no cache do navegador na solicitação HTTP
If-modified-since: <data>
- *servidor*: a resposta não contém objetos se a cópia em cache estiver atualizada:
HTTP/1.0 304 Not Modified



HTTP/2

Objetivo-chave: diminuir o atraso em solicitações HTTP de múltiplos objetos

HTTP1.1: introduziu GETs múltiplos e em pipeline sobre uma única conexão TCP

- servidor responde *em ordem* (agendamento FCFS: *first-come-first-served* – primeiro a chegar, primeiro a ser servido) para as requisições GET
- com FCFS, pequeno objeto pode ter que esperar por transmissão (**head-of-line (HOL) blocking** – bloqueio de cabeça de fila) atrás de objetos grandes(s)
- recuperação de perdas (retransmissão de segmentos TCP perdidos) paralisa a transmissão de objetos

HTTP/2

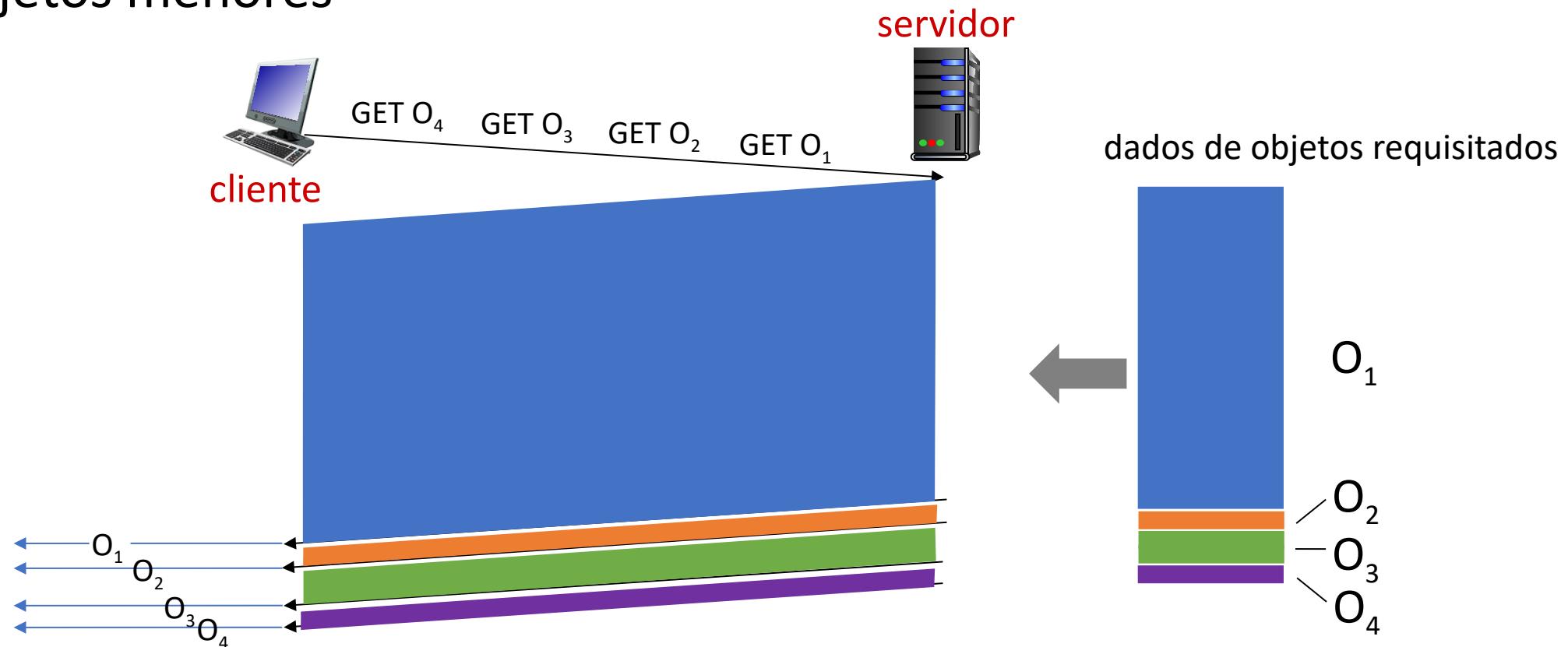
Objetivo-chave: diminuir o atraso em solicitações HTTP de múltiplos objetos

HTTP/2: [RFC 7540, 2015] maior flexibilidade no *servidor* no envio de objetos ao cliente:

- métodos, códigos de status, a maioria dos campos de cabeçalho inalterados com relação ao HTTP 1.1
- ordem de transmissão de objetos solicitados com base na prioridade de objeto especificada pelo cliente (não necessariamente FCFS)
- *empurra* objetos não requisitados para o cliente
- divide objetos em quadros, agenda quadros para mitigar o bloqueio HOL

HTTP/2: mitigando o bloqueio HOL

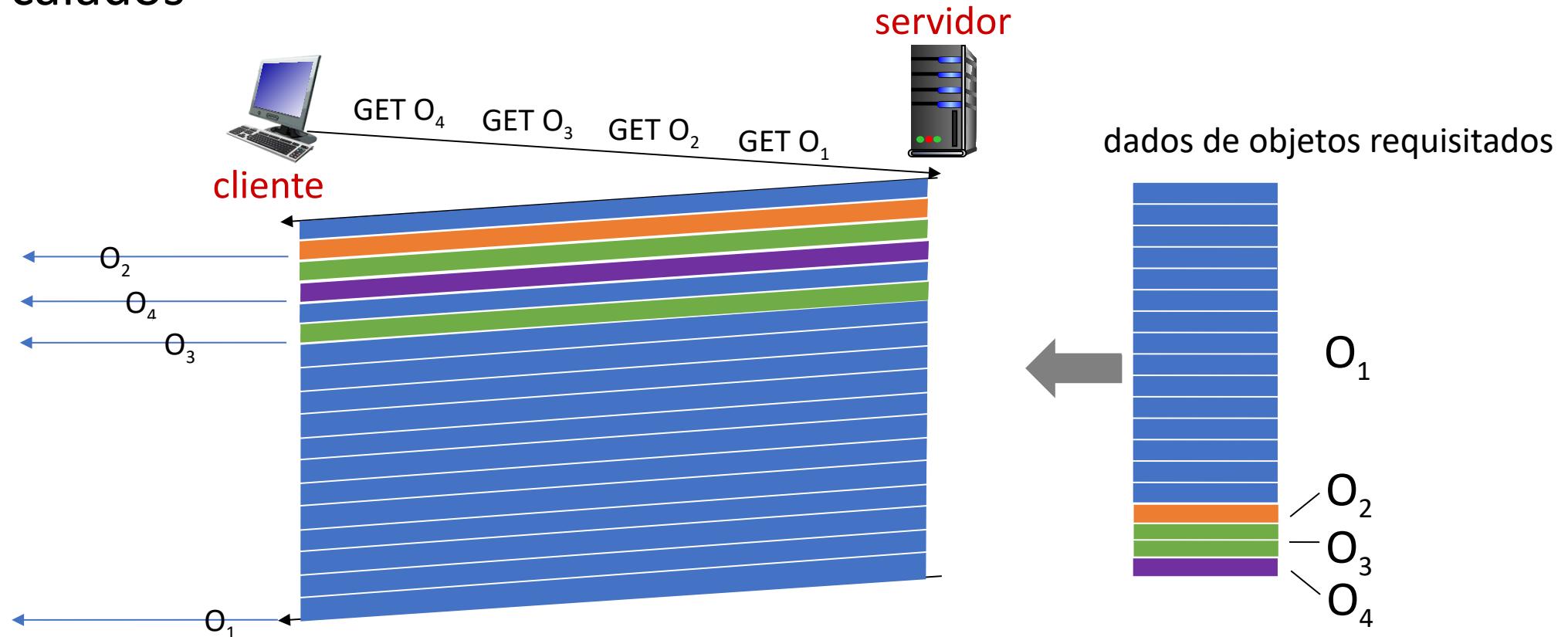
HTTP 1.1: cliente solicita 1 objeto grande (por exemplo, arquivo de vídeo) e 3 objetos menores



objetos entregues na ordem solicitada: O_2 , O_3 , O_4 esperam atrás de O_1

HTTP/2: mitigando o bloqueio HOL

HTTP/2: objetos divididos em quadros, transmissão de quadros intercalados



O_2, O_3, O_4 entregue rapidamente, O_1 levemente atrasado

HTTP/2 para HTTP/3

Rede

Versão HTTP do cliente usada



HTTP/2 sobre uma única conexão TCP significa:

- recuperção de perda de pacotes ainda paralisa todas as transmissões de objeto
 - como em HTTP 1.1, os navegadores têm incentivo para abrir várias conexões TCP paralelas para reduzir a paralisação, aumentando a vazão global
- nenhuma segurança sobre a conexão TCP tradicional
- **HTTP/3:** adiciona segurança, controle de erro e de congestionamento (mais pipeline) sobre UDP
 - mais sobre HTTP/3 na camada de transporte

Camada de aplicação: visão geral

- princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- o Domain Name System
DNS
- aplicações P2P
- transmissão de vídeo e redes de distribuição de conteúdo



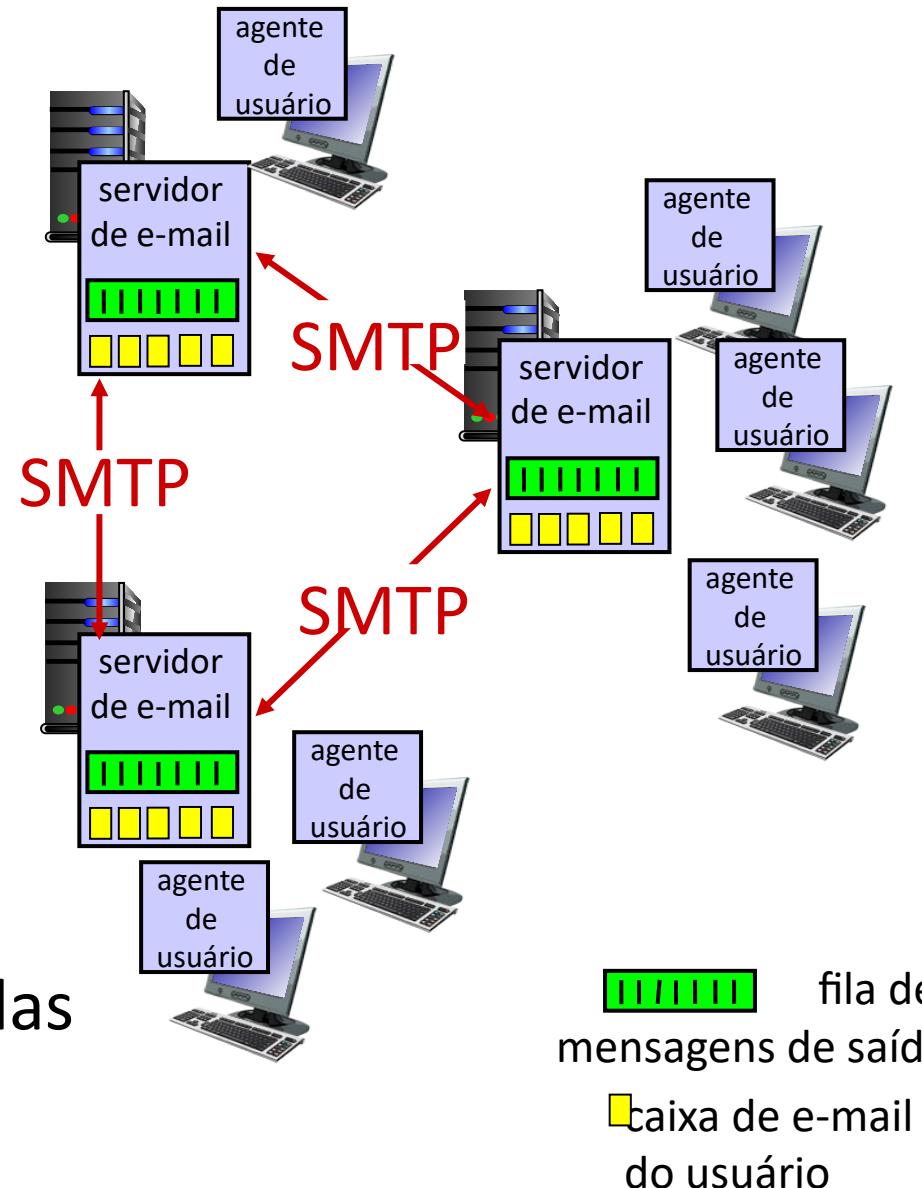
E-mail

Três componentes principais:

- agentes de usuário
- servidores de e-mail
- simple mail transfer protocol: SMTP

Agente de Usuário

- também chamado “leitor de e-mail”
- compor, editar, ler mensagens de e-mail
- ex.: Outlook, cliente de e-mail do iPhone
- mensagens entrando e saindo armazenadas no servidor



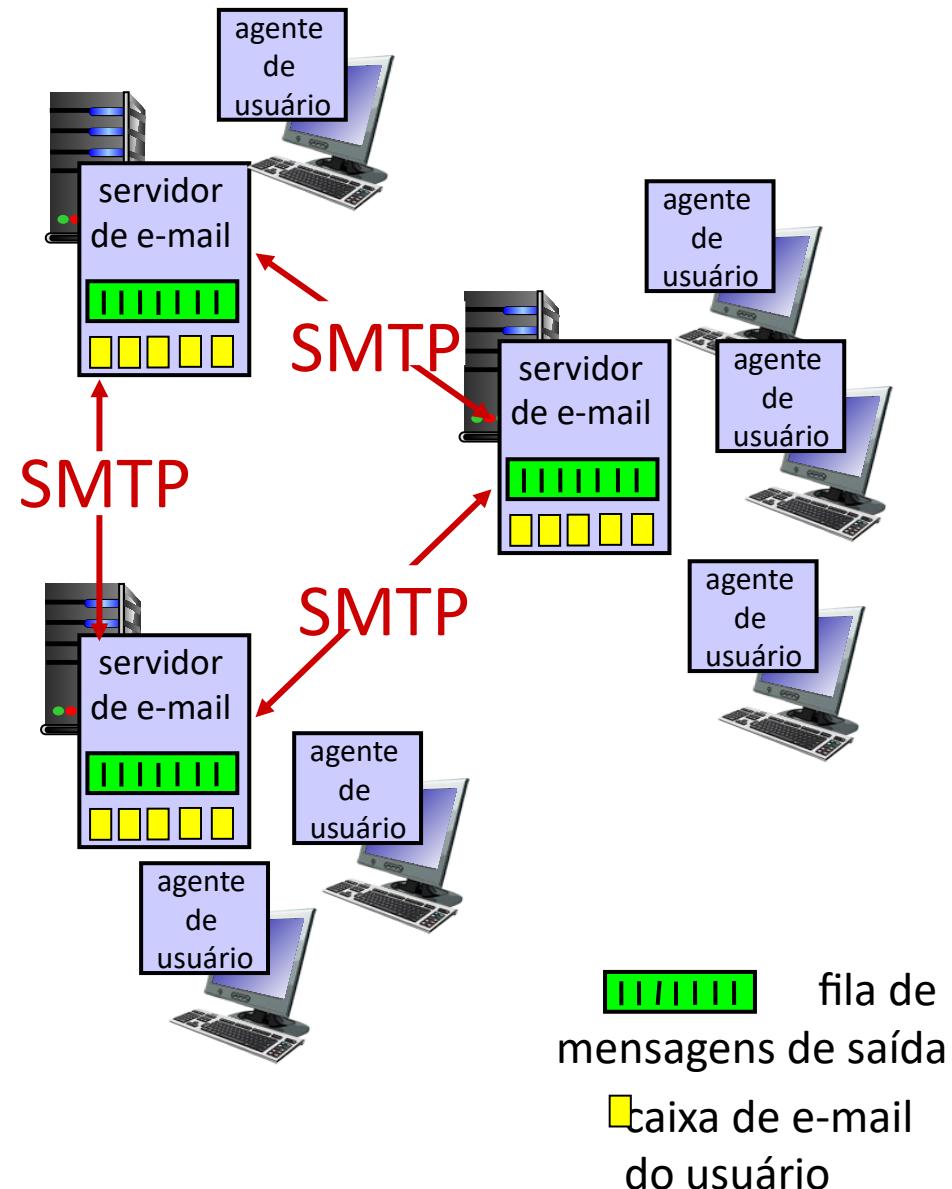
E-mail: servidores de e-mail

servidores de e-mail:

- *caixa de e-mail* contém mensagens recebidas para o usuário
- *fila de mensagens* de mensagens de e-mail de saída (a ser enviadas)

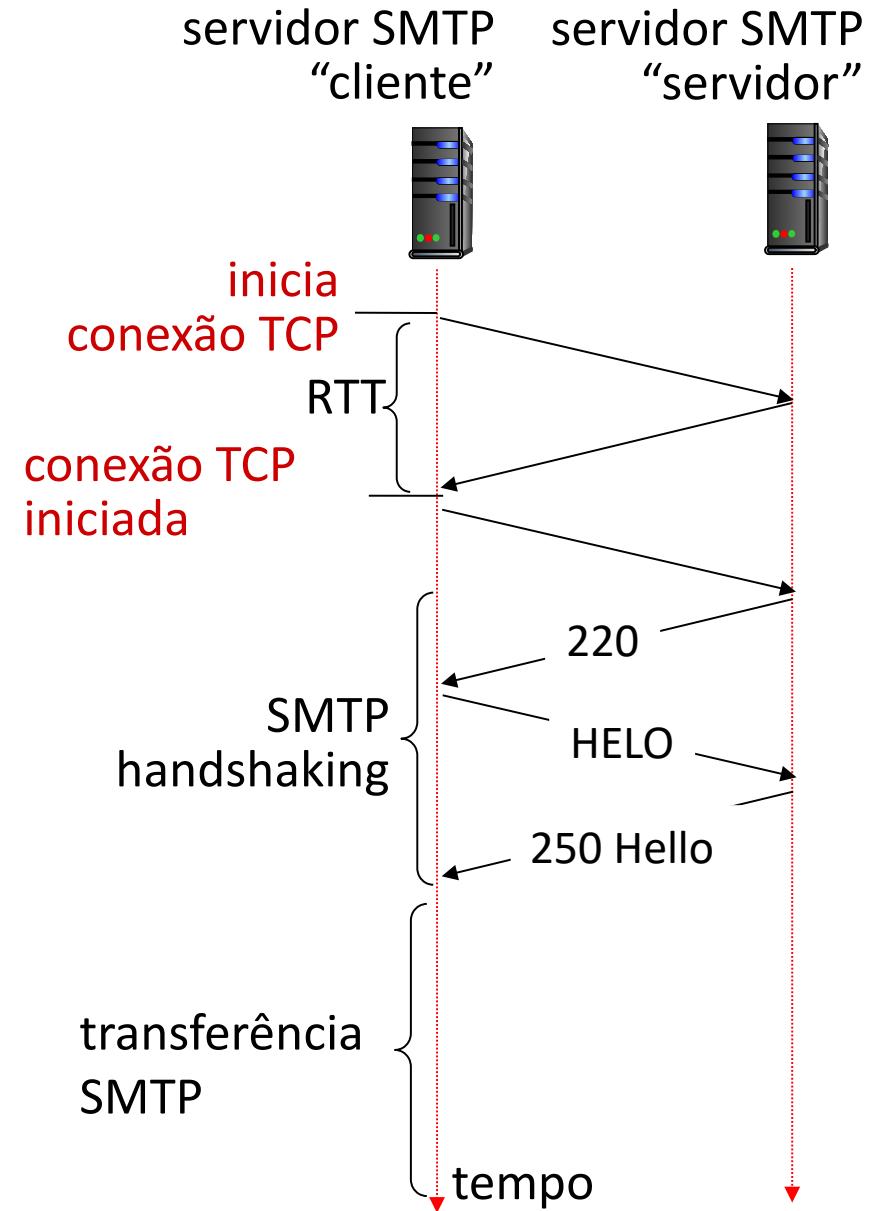
protocolo SMTP entre servidores de e-mail para enviar mensagens de e-mail

- *cliente*: servidor de e-mail enviando
- “*servidor*”: servidor de e-mail recebendo



SMTP RFC (5321)

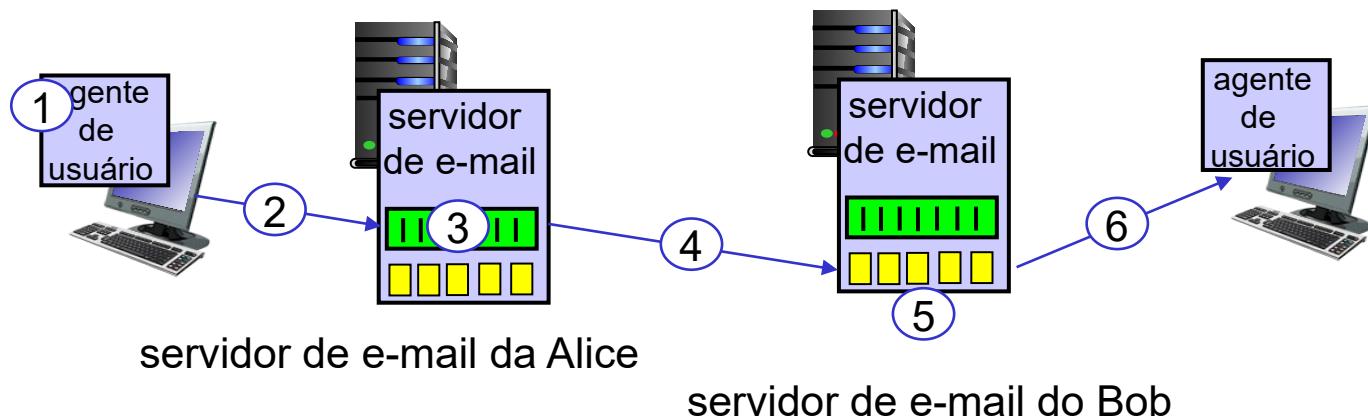
- usa o TCP para transferir de maneira confiável mensagem de e-mail do cliente (servidor de e-mail iniciando conexão) para servidor, porta 25
 - transferência direta: envio de servidor (agindo como cliente) para servidor receptor
- três fases de transferência
 - handshaking (saudação) SMTP
 - transferência de mensagens SMTP
 - fechamento do SMTP
- interação comando/resposta (como HTTP)
 - comandos: texto ASCII
 - resposta: código e frase de status



Cenário: Alice envia e-mail para o Bob

- 1) Alice usa agente de usuário para compor mensagem de e-mail “para” bob@someschool.edu
- 2) o agente de usuário de Alice envia mensagem ao seu servidor de e-mail usando SMTP; mensagem colocada na fila de mensagens
- 3) lado cliente do SMTP no servidor de e-mail abre conexão TCP com servidor de e-mail do Bob

- 4) cliente SMTP envia mensagem de Alice pela conexão TCP
- 5) o servidor de e-mail do Bob coloca a mensagem na caixa de e-mail do Bob
- 6) Bob invoca seu agente de usuário para ler mensagem



Exemplo de interação SMTP

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

SMTP: observações

comparação com HTTP:

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

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

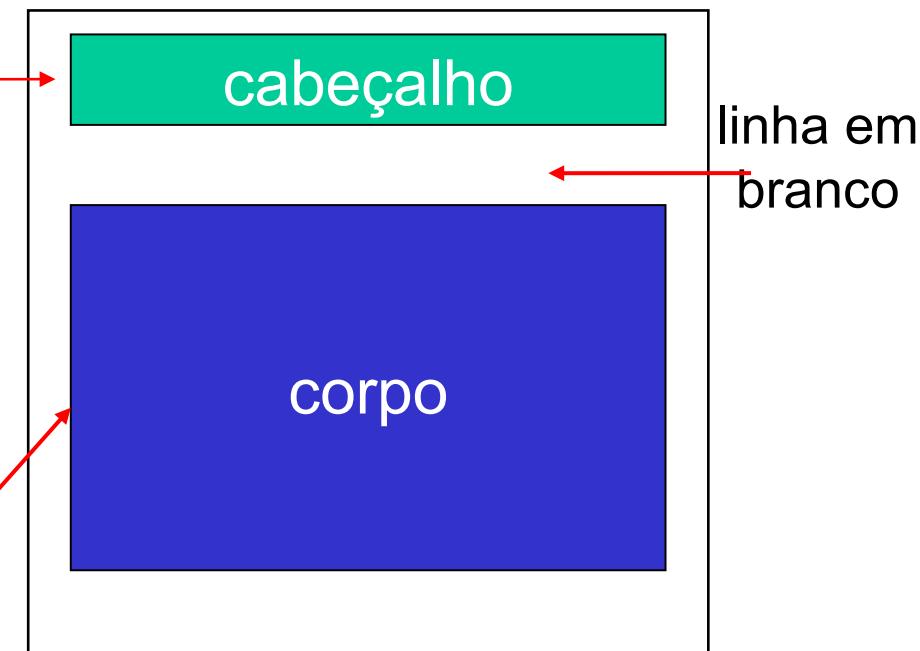
Formato de mensagem de e-mail

SMTP: protocolo para troca de mensagens de e-mail, definido na RFC 5321 (como a RFC 7231 define HTTP)

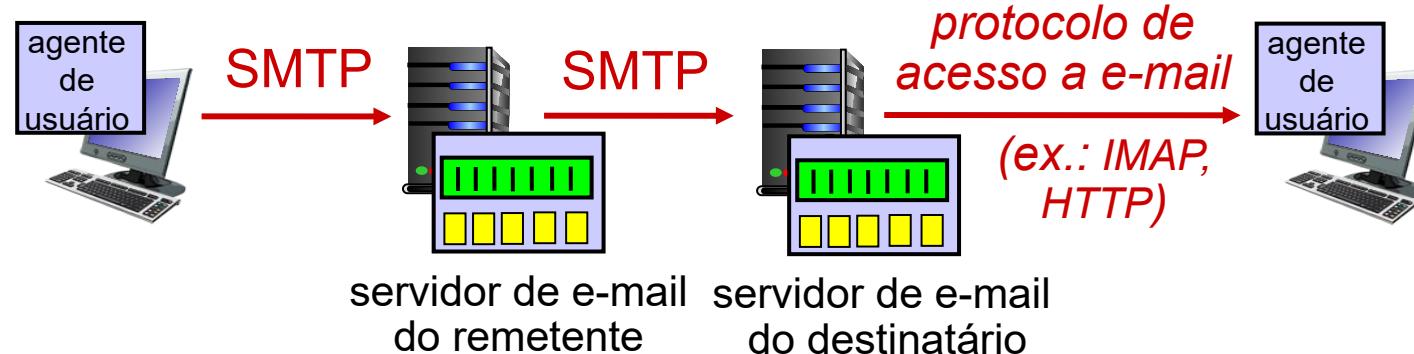
RFC 2822 define a *sintaxe* para mensagem de e-mail em si (como HTML define sintaxe para documentos da Web)

- Linhas de cabeçalho, ex.:
 - To:
 - From:
 - Subject:

essas linhas, dentro do corpo da área de mensagem de e-mail são diferentes dos comandos SMTP MAIL FROM:, RCPT TO:!
- Corpo: a “mensagem”, apenas caracteres ASCII



Recuperando e-mails: protocolos de acesso ao e-mail



- **SMTP:** entrega/armazenamento de mensagens de e-mail para o servidor do destinatário
- protocolo de acesso ao e-mail: recuperação do servidor
 - **IMAP:** Internet Mail Access Protocol [RFC 3501]: mensagens armazenadas no servidor, o IMAP fornece recuperação, exclusão, pastas de mensagens armazenadas no servidor
- **HTTP:** gmail, Hotmail, Yahoo!Mail, etc. fornece interface baseada na Web em cima de SMTP (para enviar), IMAP (ou POP) para recuperar mensagens de e-mail

Camada de aplicação: visão geral

- princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- o Domain Name System DNS
- aplicações P2P
- transmissão de vídeo e redes de distribuição de conteúdo



DNS: Domain Name System

pessoas: muitos identificadores:

- nome, RG, CPF, passaporte, título de eleitor

hospedeiros e roteadores da Internet:

- endereço IP (32 bits) - usado para endereçar datagramas
- “nome”, por exemplo, cs.umass.edu - usado por humanos
- Q: como mapear entre endereço IP e nome, e vice-versa?

Domain Name System (DNS):

- *banco de dados distribuído* implementado em uma hierarquia de muitos *servidores de nome*
- *protocolo de camada de aplicação:* hospedeiros e servidores DNS se comunicam para *resolver* nomes (tradução endereço/nome)
 - *nota:* função básica da Internet, **implementada como protocolo de camada de aplicação**
 - complexidade na “borda” da rede

DNS: serviços, estrutura

serviços de DNS:

- tradução de nome de hospedeiro para IP
- apelidos de hospedeiro
 - nomes canônicos e alternativos
- apelido do servidor de e-mail
- distribuição de carga
 - servidores Web replicados: muitos endereços IP correspondem a um nome

Q: Por que não centralizar o DNS?

- único ponto de falha
- volume de tráfego
- banco de dados centralizado distante
- manutenção

R: não escala!

- Somente servidores de DNS da Comcast: 600 bilhões de consultas DNS / dia
- Somente servidores de DNS da Akamai: 2,2 trilhões de consultas DNS / dia

Pensando no DNS

enorme banco de dados distribuído:

- ~ bilhões de registros

lida com muitos *trilhões* de consultas/dia:

- *muito* mais leituras que escritas
- *desempenho importa*: quase todas as transações de Internet interagem com DNS
 - milissegundos contam!

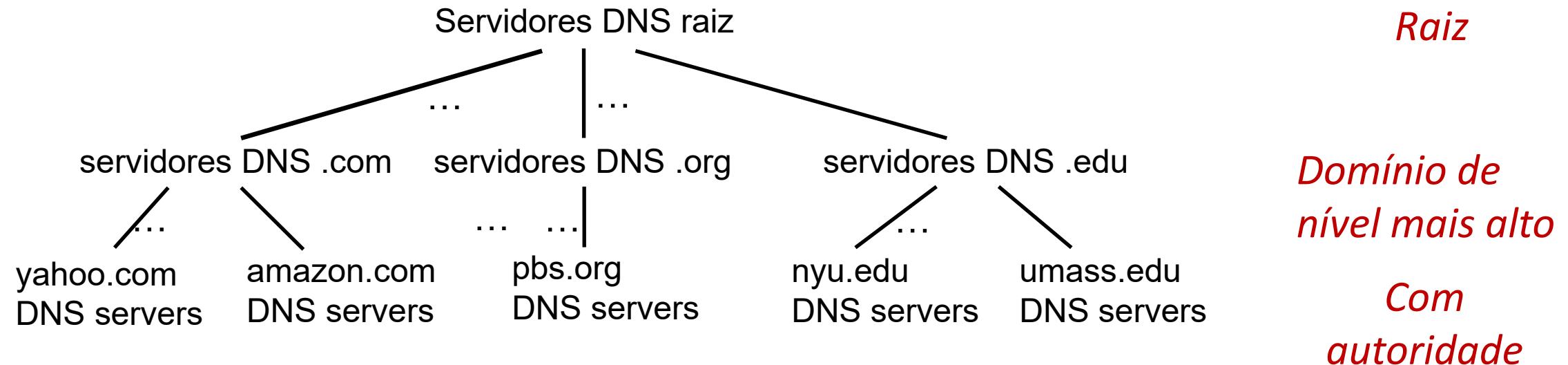
Organizacionalmente e fisicamente descentralizado:

- milhões de organizações diferentes responsáveis por seus registros

“a prova de balas”: confiabilidade, segurança



DNS: um banco de dados distribuído e hierárquico

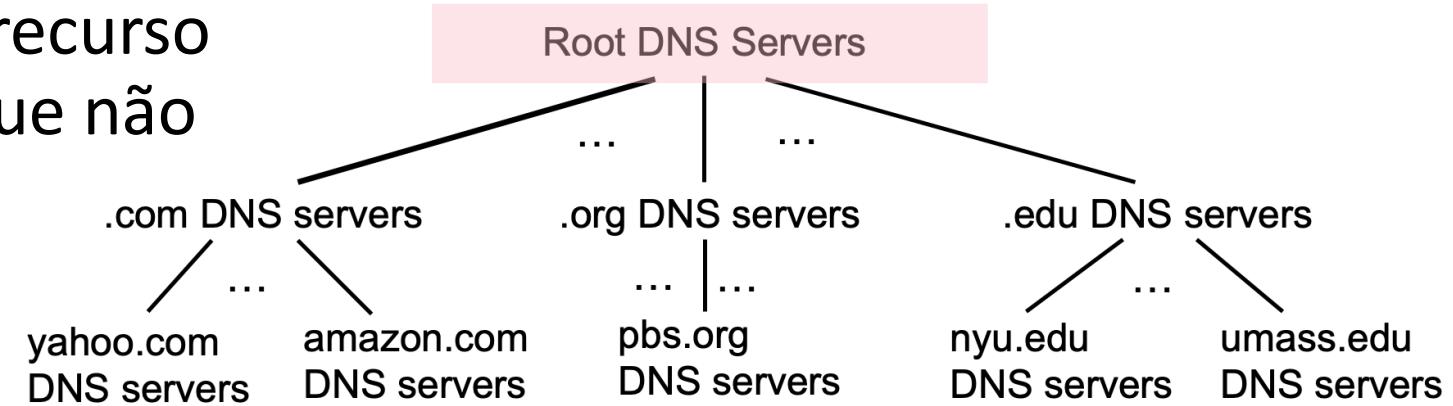


O cliente quer endereço IP para www.amazon.com; 1º contato:

- cliente consulta servidor raiz para encontrar servidor DNS de .com
- cliente consulta servidor DNS de .com para obter o servidor DNS de amazon.com
- cliente consulta servidor DNS de amazon.com para obter o endereço IP de www.amazon.com

DNS: servidores de nome raiz

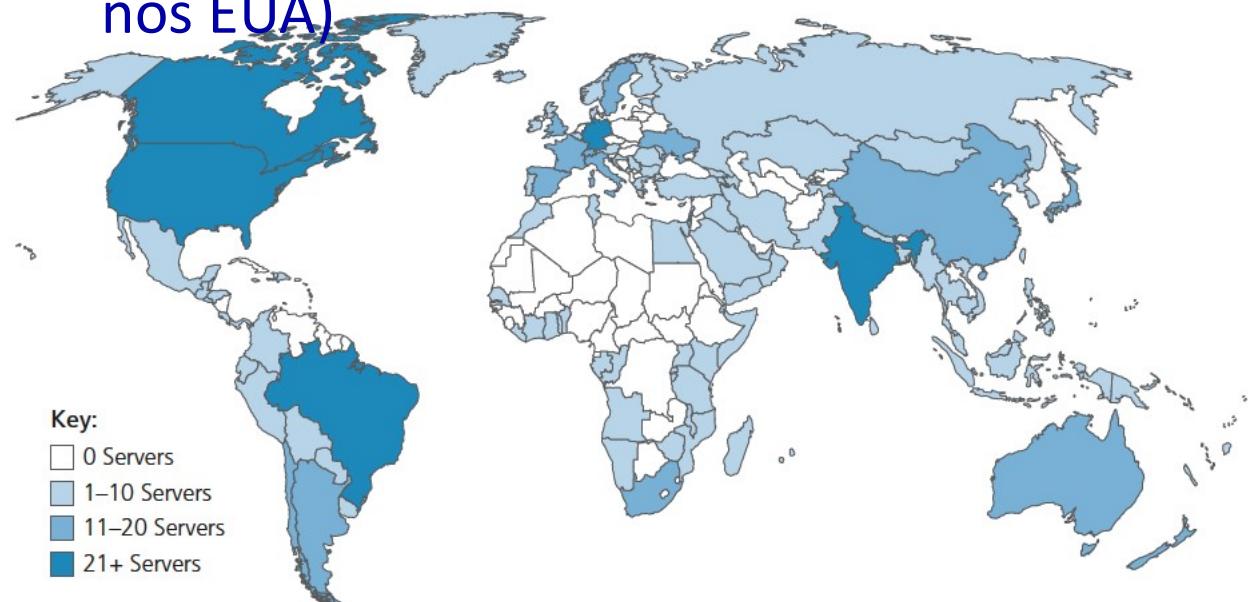
- oficial, contato de último recurso por servidores de nome que não podem resolver um nome



DNS: servidores de nome raiz

- oficial, contato de último recurso por servidores de nome que não podem resolver um nome
- função da Internet *incrivelmente importante*
 - Internet não poderia funcionar sem ele!
 - DNSSEC – fornece segurança (autenticação, integridade de mensagem)
- ICANN (Internet Corporation for Assigned Names and Numbers) gerencia domínio DNS raiz

13 “servidores” de nome raiz lógicos em todo o mundo, cada “servidor” replicado muitas vezes (~200 servidores nos EUA)



News and publications [show all](#)

2024-04-15 [Root Server System Operational Information](#)

2022-08-02 [Statement on adding ZONEMD to the root zone](#)

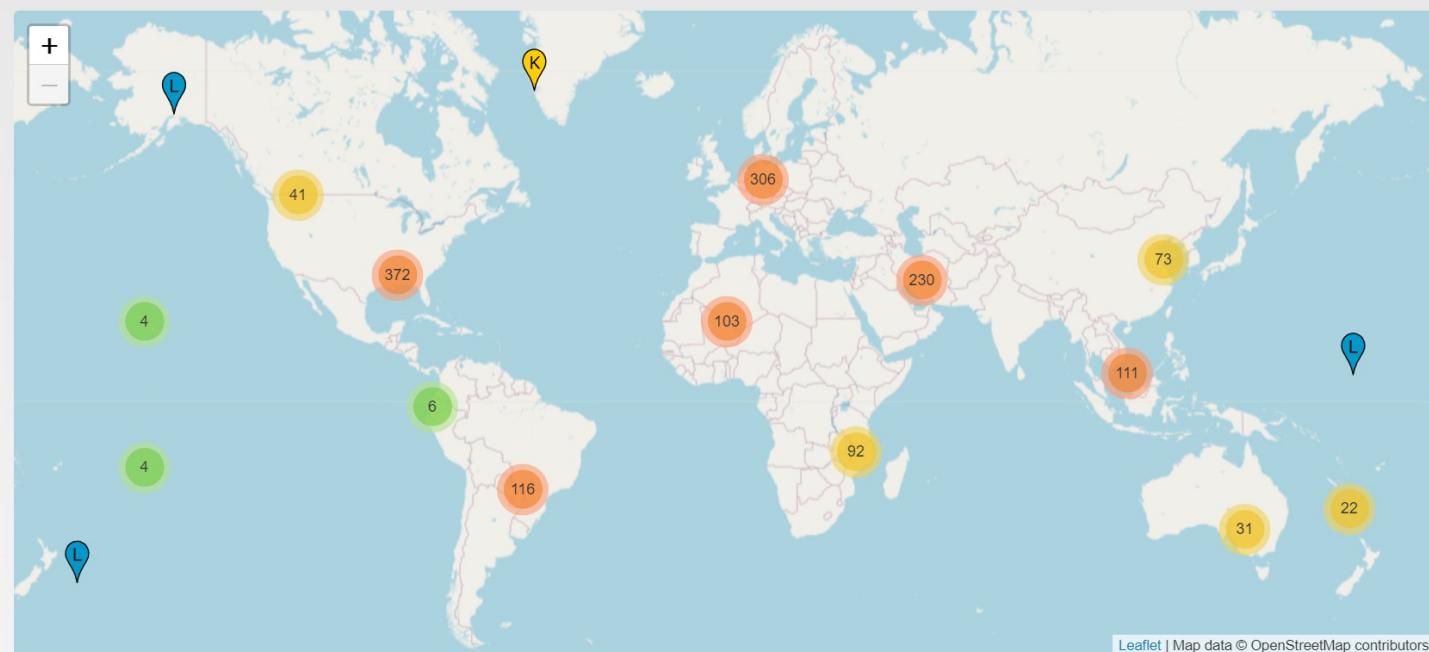
2021-03-30 [Statement on DNS Encryption](#)

Meeting agendas [show all](#)

2024-07-21 [IETF 120/Vancouver/Virtual \(PDF\)](#)

2024-03-17 [IETF 119/Brisbane/Virtual \(PDF\)](#)

2023-11-05 [IETF 118/Prague/Virtual \(PDF\)](#)



The 13 root name servers are operated by 12 independent organisations.

You can find more information about each of these organisations by visiting their homepage as found in the 'Operator' field below.

Technical questions about the Root Server System as a whole can be directed to the [Ask RSSAC](#) e-mail address.

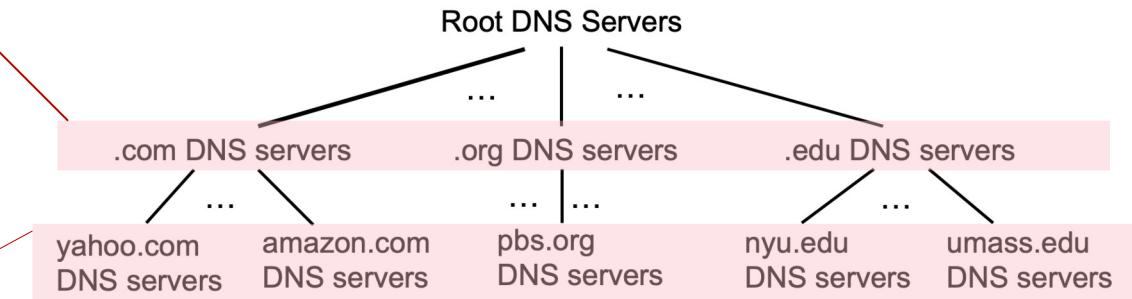
Visualisations produced from RSSAC002 data submitted by the root server operators can be viewed at [rssac002.root-servers.org](#)

<http://www.root-servers.org/>

Domínio de nível mais alto, e servidores com autoridade

servidores Top-Level Domain (TLD – domínio de nível mais alto):

- responsáveis por .com, .org, .net, .edu, .aero, .jobs, .museums, e todos os domínios de países de nível mais alto, ex.: .cn, .uk, .fr, .ca, .jp, .br
- Network Solutions: registro com autoridade para TLDs .com, .net
- Educause: TLD .edu



servidores DNS com autoridade:

- servidores DNS da própria organização, fornecendo mapeamentos de nome de hospedeiro para IP com autoridade para os hospedeiros nomeados da organização
- pode ser mantido pela organização ou provedor de serviços

Root Zone Database

Domain Names
Overview
Root Zone Management
Overview
Root Database
Hint and Zone Files
Change Requests
Instructions & Guides
Root Servers
.INT Registry
.ARPA Registry
IDN Practices Repository
Root Key Signing Key (DNSSEC)
Reserved Domains

The Root Zone Database represents the delegation details of top-level domains, including gTLDs such as .com, and country-code TLDs such as .uk. As the manager of the DNS root zone, we are responsible for coordinating these delegations in accordance with our [policies and procedures](#).

Much of this data is also available via the WHOIS protocol at whois.iana.org.

DOMAIN	TYPE	TLD MANAGER
.aaa	generic	American Automobile Association, Inc.
.aarp	generic	AARP
.abarth	generic	Not assigned
.abb	generic	ABB Ltd
.abbott	generic	Abbott Laboratories, Inc.
.abbvie	generic	AbbVie Inc.
.abc	generic	Disney Enterprises, Inc.
.able	generic	Able Inc.
.abogado	generic	Registry Services, LLC
.abudhabi	generic	Abu Dhabi Systems and Information Centre
.ac	country-code	Internet Computer Bureau Limited
.academy	generic	Binky Moon, LLC
.accenture	generic	Accenture plc
.accountant	generic	dot Accountant Limited
.accountants	generic	Binky Moon, LLC
.aco	generic	ACO Severin Ahlmann GmbH & Co. KG
.active	generic	Not assigned
.actor	generic	Dog Beach, LLC
.ad	country-code	Andorra Telecom
.adac	generic	Not assigned
.ads	generic	Charleston Road Registry Inc.
.adult	generic	ICM Registry AD LLC
.ae	country-code	Telecommunications and Digital Government Regulatory Authority (TDRA)
.aeg	generic	Aktiebolaget Electrolux
.aero	sponsored	Societe Internationale de Telecommunications Aeronautique (SITA INC USA)
.aetna	generic	Aetna Life Insurance Company
.af	country-code	Ministry of Communications and IT
.afamilycompany	generic	Not assigned
.afl	generic	Australian Football League

Servidores DNS locais

- quando o hospedeiro faz uma consulta DNS, ela é enviada para o seu servidor DNS *local*
 - Servidor DNS local retorna, respondendo:
 - de seu cache local de pares de tradução nome-endereço recentes (possivelmente desatualizado!)
 - encaminhando requisições para a hierarquia de DNS para resolução
 - cada ISP tem servidor DNS local; para encontrar o seu:
 - MacOS: % scutil --dns
 - Windows: >ipconfig /all
- servidor DNS local não pertence estritamente à hierarquia

```
PS C:\Users\fbrev> ipconfig /all
```

Configuração de IP do Windows

Nome do host. : DONALD
Sufixo DNS primário :
Tipo de nó. : híbrido
Roteamento de IP ativado. : não
Proxy WINS ativado. : não

Adaptador Ethernet vEthernet (Ethernet Intel do Host):

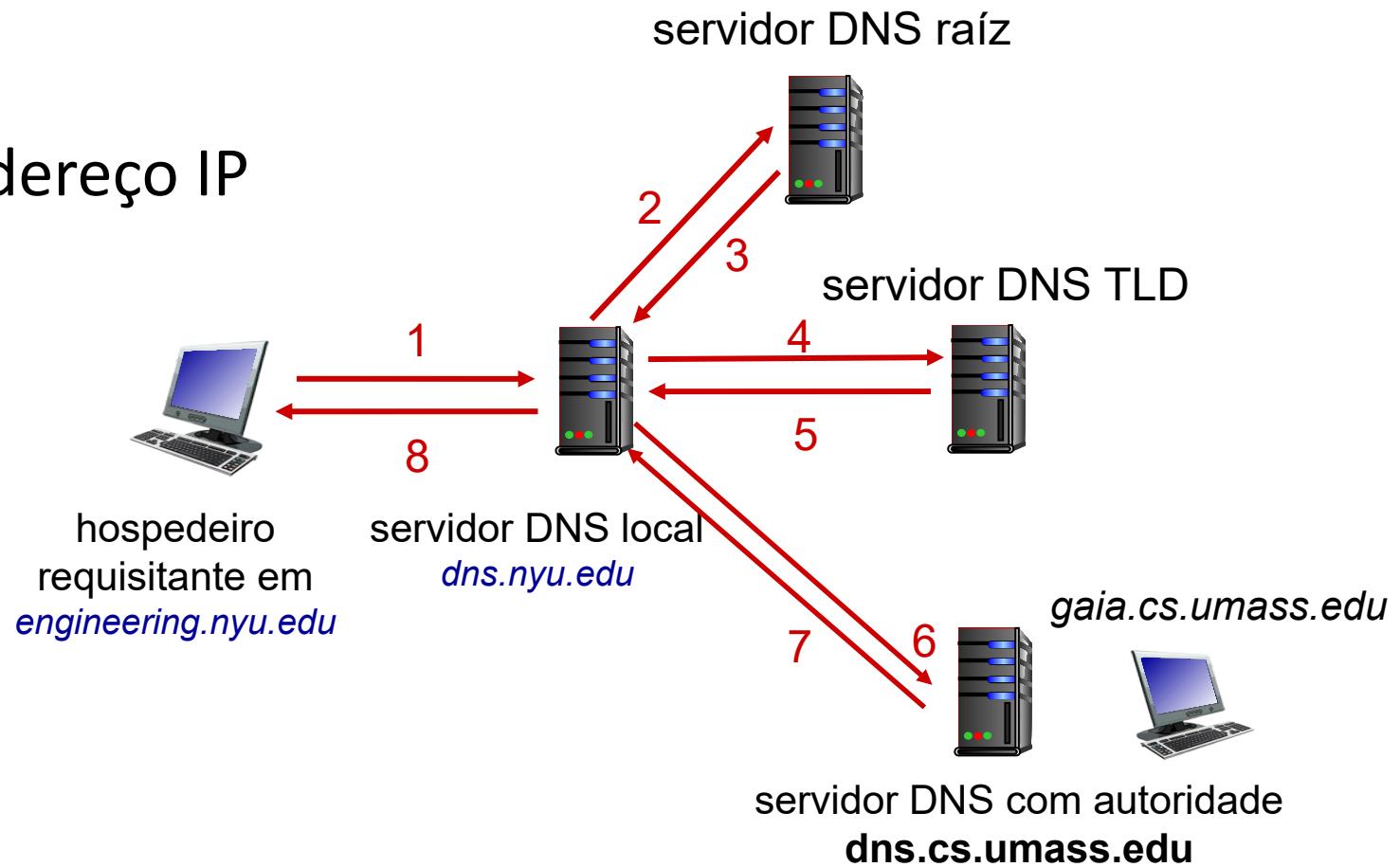
Sufixo DNS específico de conexão. :
Descrição : Hyper-V Virtual Ethernet Adapter #3
Endereço Físico : 08-BF-B8-37-D3-CC
DHCP Habilitado : Sim
Configuração Automática Habilitada. : Sim
Endereço IPv6 : 2804:7f0:9b80:210e:1cf4:cb07:5e4f:32c6(Preferencial)
Endereço IPv6 Temporário. : 2804:7f0:9b80:210e:90a4:39a5:f01e:a1e5(Preferencial)
Endereço IPv6 de link local : fe80::76c4:fa00:ddfe:1ecb%19(Preferencial)
Endereço IPv4. : 192.168.50.5(Preferencial)
Máscara de Sub-rede : 255.255.255.0
Concessão Obtida. : sexta-feira, 23 de agosto de 2024 13:25:42
Concessão Expira. : sábado, 24 de agosto de 2024 13:25:42
Gateway Padrão. : fe80::a236:bcff:feaf:f5c0%19
192.168.50.1
Servidor DHCP : 192.168.50.1
IAID de DHCPv6. : 470335416
DUID de Cliente DHCPv6. : 00-01-00-01-2E-56-71-5C-08-BF-B8-37-D3-CC
Servidores DNS. : 2804:7f0:9b80:210e::1
192.168.50.1
NetBIOS em Tcpip. : Habililtado

Resolução de nomes do DNS: consulta iterativa

Exemplo: hospedeiro em engineering.nyu.edu quer o endereço IP para gaia.cs.umass.edu

Consulta iterativa:

- servidor contatado responde com o nome do servidor a contatar
- “Eu não sei esse nome, mas pergunte a este servidor”

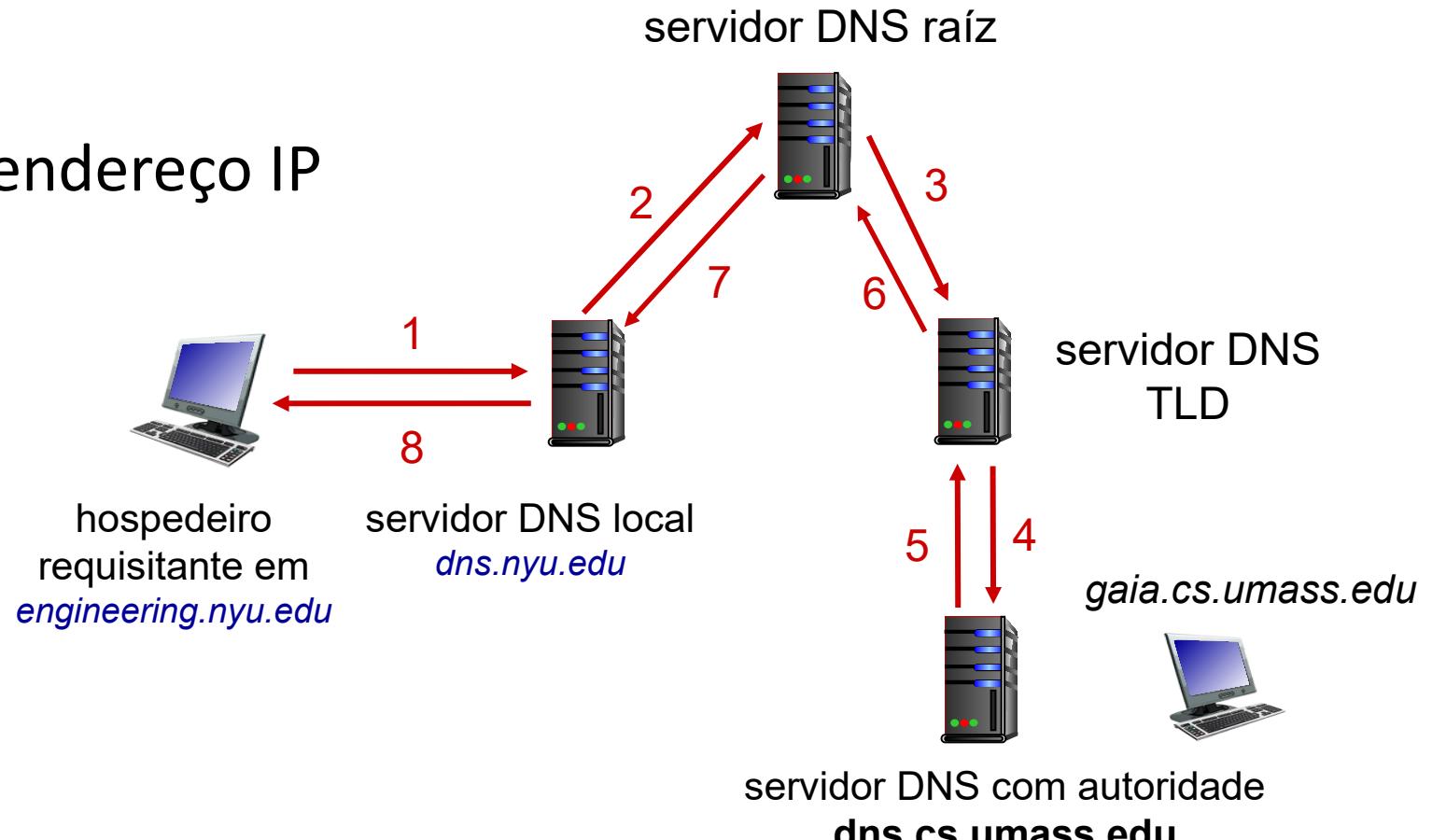


Resolução de nomes do DNS: consulta recursiva

Exemplo: hospedeiro em engineering.nyu.edu quer o endereço IP para gaia.cs.umass.edu

Consulta recursiva:

- coloca o fardo da resolução de nomes no servidor de nomes contatado
- carga pesada em níveis superiores da hierarquia?



Cache de Informações de DNS

- uma vez que (qualquer) servidor de nome aprende um mapeamento, ele armazena esse mapeamento em um *cache*, e *imediatamente* retorna o mapeamento armazenado quando recebe uma nova requisição
 - cache melhora o tempo de resposta
 - entradas no cache expiram (desaparecem) após algum tempo (TTL)
 - Servidores TLD normalmente estão armazenados em cache em servidores de nomes locais
- entradas em cache podem estar *desatualizadas*
 - se o hospedeiro nomeado mudar de endereço IP, pode não ser acessível em toda a Internet até que todos os TTLs expirem!
 - *tradução de nome para endereço de melhor esforço!*

Registros DNS

DNS: banco de dados distribuído armazenando registros de recursos (RR)

Formato RR: (nome, valor, tipo, ttl)

tipo=A

- nome é nome de hospedeiro
- valor é endereço IP

tipo=NS

- nome é domínio (ex.: foo.com)
- valor é nome de hospedeiro de um servidor de nomes com autoridade para este domínio

tipo=CNAME

- nome é um apelido para algum nome “canônico” (o nome real)
- www.ibm.com é na verdade servereast.backup2.ibm.com
- valor é o nome canônico

tipo=MX

- valor é o nome de um servidor de SMTP (e-mail) associado com nome

Colocando suas informações no DNS

exemplo: nova startup “Network Utopia”

- registre o nome networkuptopia.com no *Registrador de DNS* (ex.: Network Solutions)
- forneça nomes e endereços IP dos servidores de nome com autoridade (principal e secundário)
 - registrador insere registros NS e A no servidor TLD .com:
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
- crie servidor com autoridade localmente com o endereço IP 212.212.212.1
 - registro tipo A para www.networkuptopia.com
 - registro tipo MX para networkutopia.com

Windows PowerShell

X + | v

- □ X

```
PS C:\Users\fbrev> nslookup -type=NS fabriciobreve.com
```

```
Servidor: RT-AX86U-F5C0
```

```
Address: 2804:7f0:9b80:210e::1
```

Não é resposta autoritativa:

```
fabriciobreve.com      nameserver = vera.ns.cloudflare.com
```

```
fabriciobreve.com      nameserver = aragorn.ns.cloudflare.com
```

```
PS C:\Users\fbrev> nslookup www.fabriciobreve.com
```

```
Servidor: RT-AX86U-F5C0
```

```
Address: 2804:7f0:9b80:210e::1
```

Não é resposta autoritativa:

```
Nome: www.fabriciobreve.com
```

```
Addresses: 2606:4700:3030::ac43:8377
```

```
2606:4700:3034::6815:a72
```

```
104.21.10.114
```

```
172.67.131.119
```

```
PS C:\Users\fbrev> nslookup -type=MX fabriciobreve.com
```

```
Servidor: RT-AX86U-F5C0
```

```
Address: 2804:7f0:9b80:210e::1
```

Não é resposta autoritativa:

```
fabriciobreve.com      MX preference = 1, mail exchanger = aspmx.l.google.com
```

```
fabriciobreve.com      MX preference = 5, mail exchanger = alt1.aspmx.l.google.com
```

```
fabriciobreve.com      MX preference = 5, mail exchanger = alt2.aspmx.l.google.com
```

```
fabriciobreve.com      MX preference = 10, mail exchanger = aspmx2.googlemail.com
```

```
fabriciobreve.com      MX preference = 10, mail exchanger = aspmx3.googlemail.com
```

```
PS C:\Users\fbrev> |
```

Segurança do DNS

Ataques DDoS

- bombardear servidores raiz com tráfego
 - sem sucesso até hoje
 - filtragem de tráfego
 - servidores DNS locais fazem cache dos IPs de servidores TLD, permitindo contornar os servidores raiz
- bombardear servidores TLD
 - potencialmente mais perigoso

Ataques de spoofing

- interceptar consultas DNS, retornando respostas falsas
 - envenenamento de cache de DNS
 - RFC 4033: serviços de autenticação DNSSEC

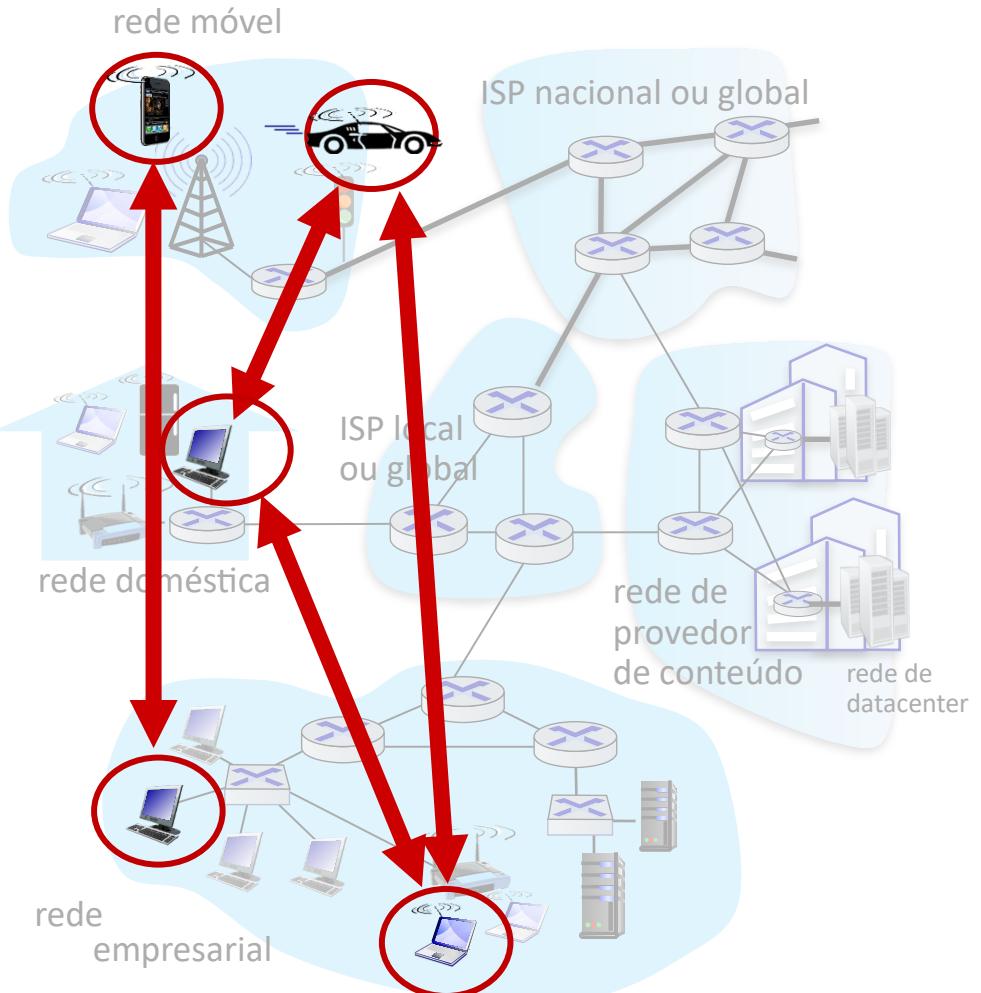
Camada de aplicação: visão geral

- princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- o Domain Name System
DNS
- aplicações P2P
- transmissão de vídeo e redes de distribuição de conteúdo



Arquitetura Peer-to-peer (P2P)

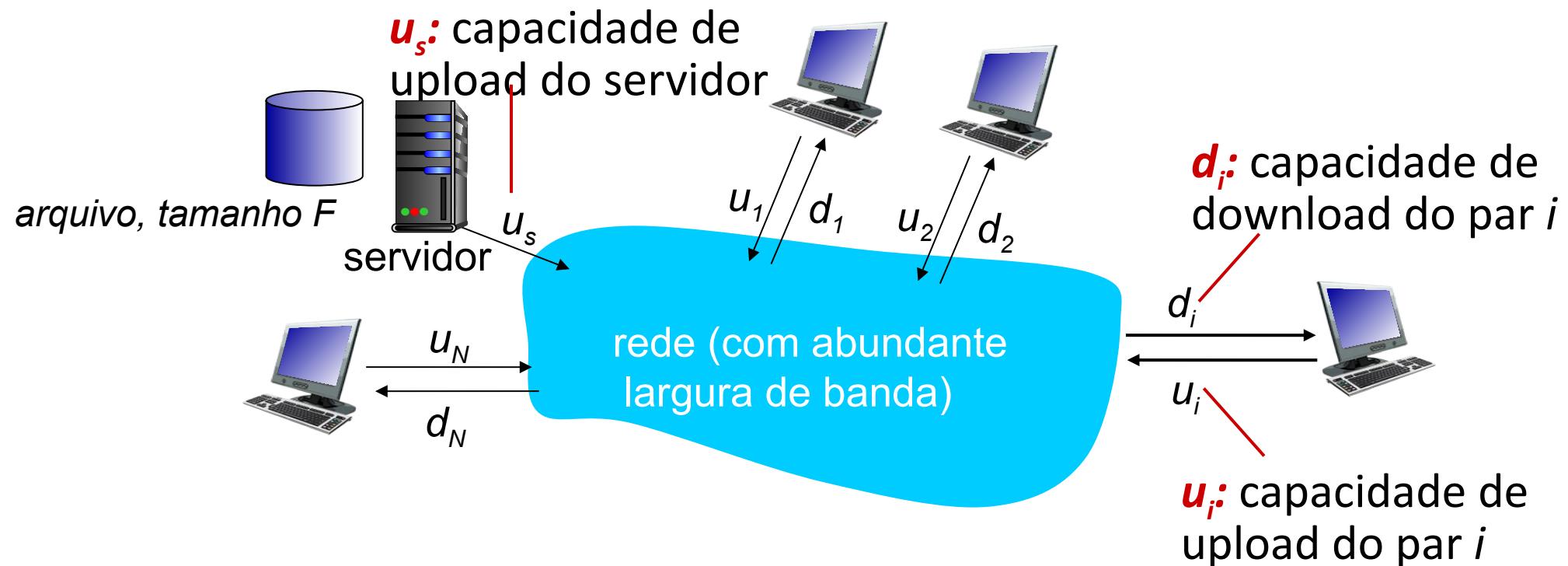
- *sem servidor sempre ligado*
- sistemas finais arbitrários se comunicam diretamente
- pares solicitam serviço de outros pares, e em troca oferecem serviços para outros pares
 - *auto escalabilidade* – novos pares trazem nova capacidade de atendimento e novas demandas de serviços
- os pares estão intermitentemente conectados e alteram endereços IP
 - gerenciamento complexo
- exemplos: Compartilhamento de arquivos P2P (BitTorrent), streaming (KanKan), VoIP (Skype)



Distribuição de arquivo: cliente-servidor vs P2P

Q: quanto tempo para distribuir arquivo (tamanho F) de um servidor para N pares?

capacidade de upload/download por pares é recurso limitado



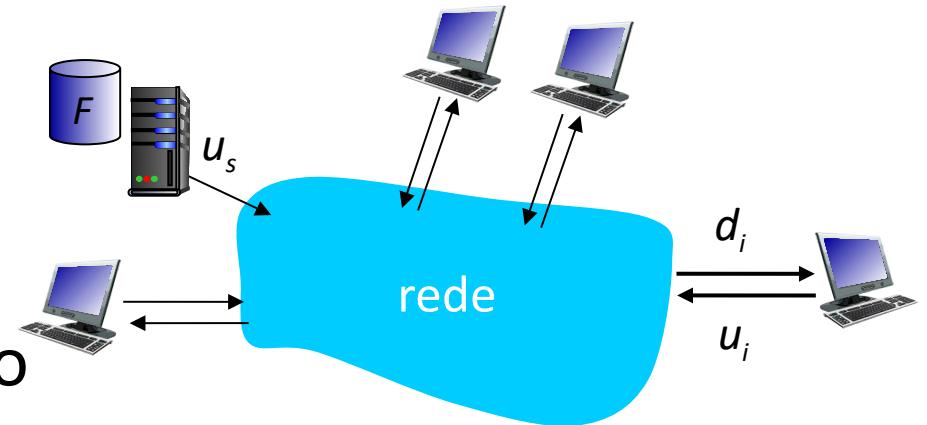
Tempo de distribuição de arquivos: cliente-servidor

- *transmissão do servidor:* deve enviar sequencialmente (upload) N cópias do arquivo:

- tempo para enviar uma cópia: F/u_s
- tempo para enviar N cópias: NF/u_s

- *cliente:* cada cliente deve baixar cópia do arquivo

- d_{min} = taxa de download mínima do cliente
- tempo de download mínimo do cliente: F/d_{min}



*tempo para distribuir F
para N clientes usando
abordagem cliente-servidor*

$$D_{c-s} > \max\{NF/u_s, F/d_{min}\}$$

aumenta linearmente em N

Tempo de distribuição de arquivo: P2P

- *transmissão do servidor*: deve enviar pelo menos uma cópia:

- tempo para enviar uma cópia: F/u_s

- *cliente*: cada cliente deve baixar cópia do arquivo

- tempo de download mínimo do cliente: F/d_{min}

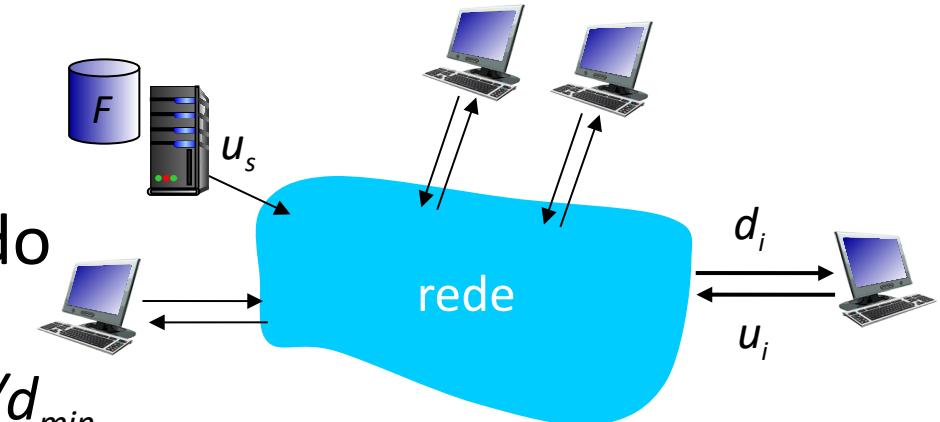
- *clientes*: agregados devem baixar NF bits
 - taxa de upload máxima (limitando a taxa de download máxima) é $u_s + \sum u_i$

tempo para distribuir F
para os N clientes
usando abordagem P2P

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

aumenta linearmente em N ...

... mas este também aumenta, pois cada par traz capacidade de serviço

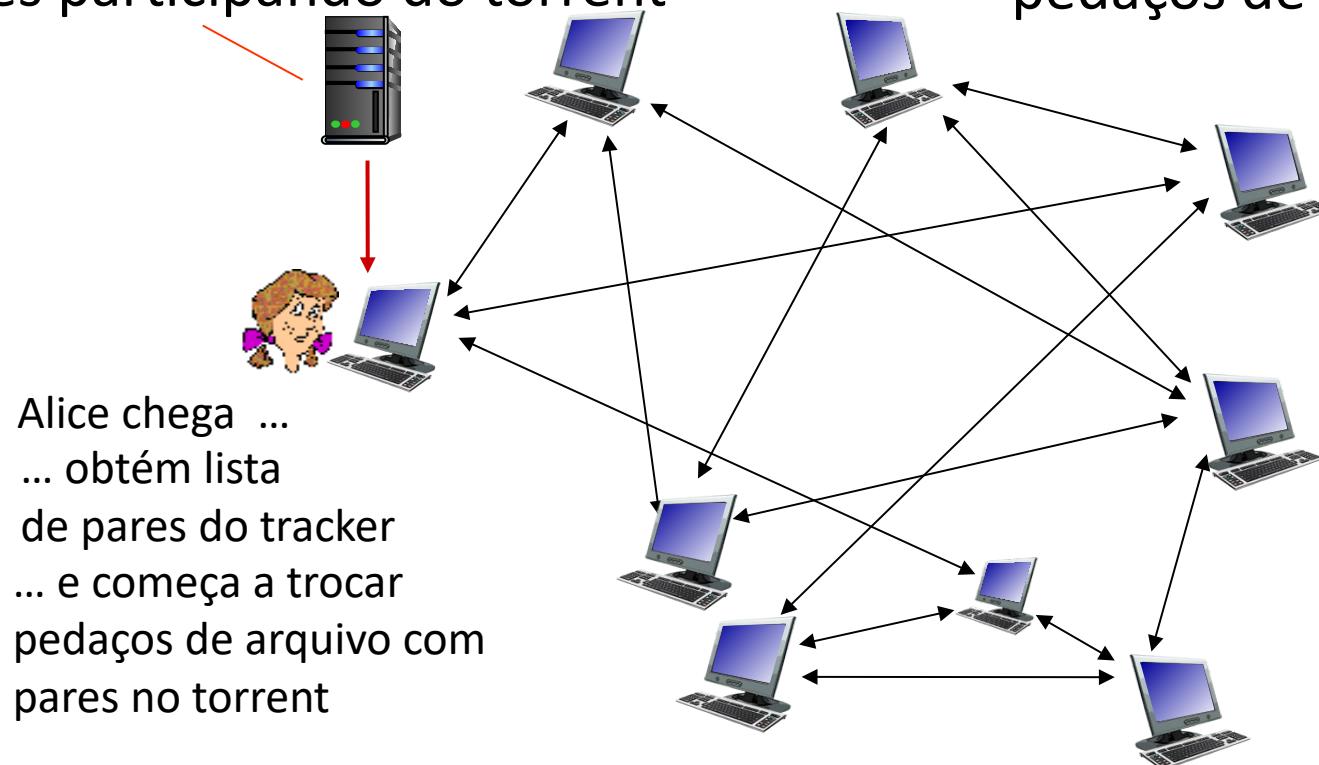


Distribuição de arquivos P2P: BitTorrent

arquivo dividido em pedaços de 256Kb

- pares em torrent enviam/recebem pedaços de arquivo

tracker (rastreador): rastreia
pares participando do torrent



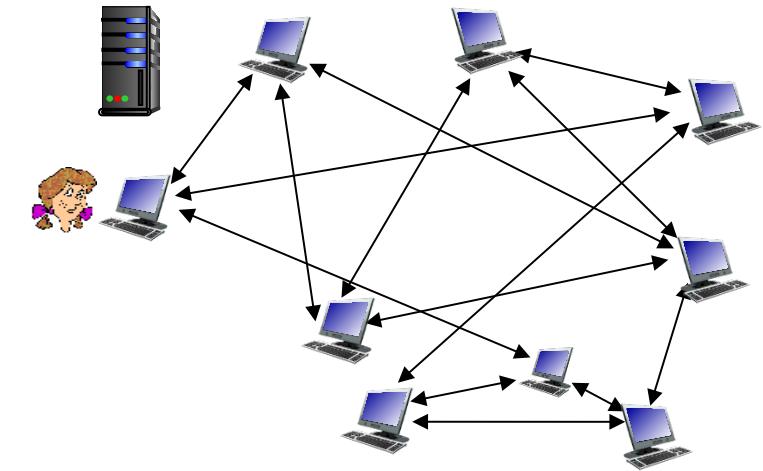
torrent: grupo de pares trocando
pedaços de um arquivo

Distribuição de arquivos P2P: BitTorrent

- par se juntando ao torrent:
- não tem pedaços, mas vai acumulá-los ao longo do tempo de outros pares
 - registra-se com tracker para obter lista de pares, conecta-se a um subconjunto de pares (“vizinhos”)

durante o download, par envia pedaços para outros pares
par pode mudar de pares com quem troca pedaços

- *churn (rotatividade)*: pares podem ir e vir
- uma vez que um par tem o arquivo inteiro, ele pode sair (sendo egoísta) ou permanecer no torrent (sendo altruísta)



BitTorrent: solicitando, enviando pedaços de arquivo

Solicitando pedaços:

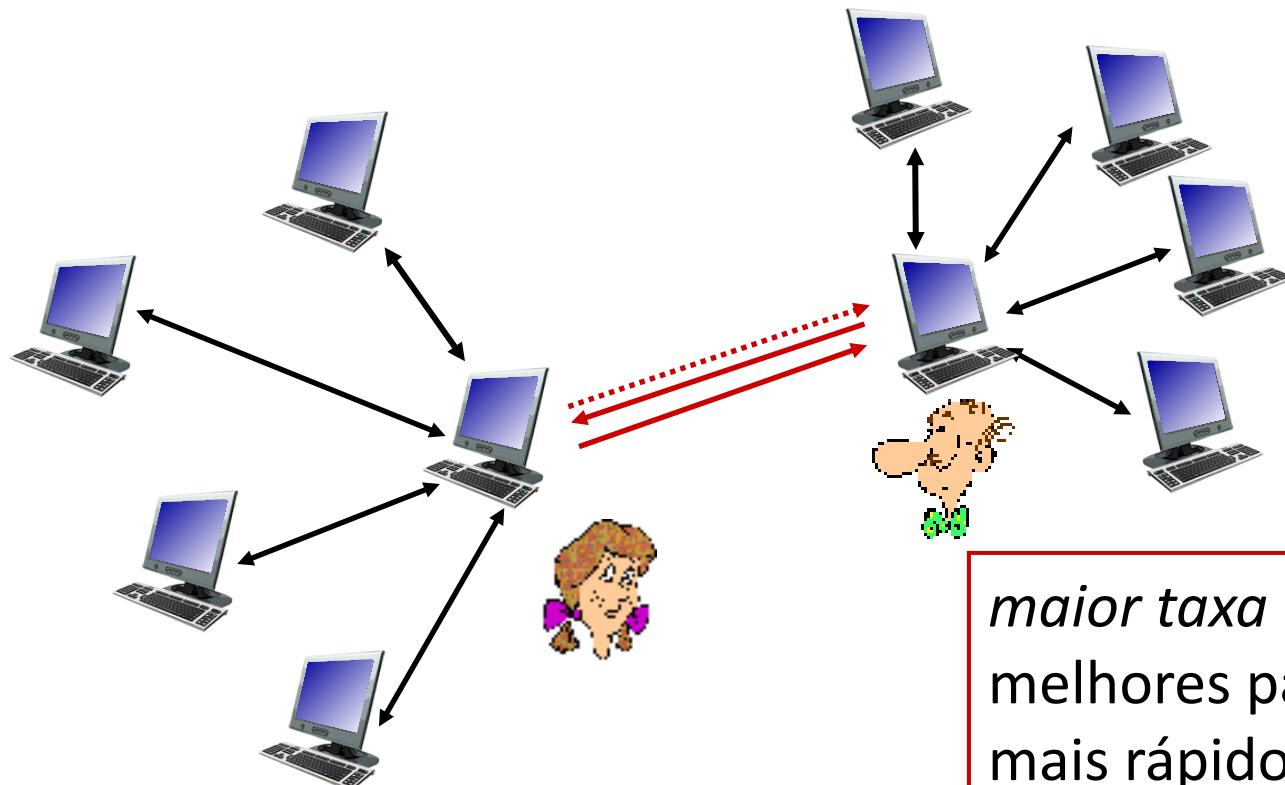
- a qualquer momento, diferentes pares têm subconjuntos diferentes de pedaços de arquivo
- periodicamente, Alice pede a cada par para que enviem uma lista dos pedaços que eles têm
- Alice pede pedaços faltantes para os pares, os mais raros primeiro

Enviando pedaços: olho por olho

- Alice envia pedaços para os quatro pares que atualmente estão enviando seus pedaços *a uma taxa mais alta*
- outros pares são sufocados por Alice (não recebem pedaços dela)
 - reavalia quem são os 4 melhores a cada 10 segundos
- a cada 30 segundos: seleciona aleatoriamente outro par, começa a enviar pedaços
 - “desafoga” este par de forma “otimista”
 - par recém-escolhido pode se juntar aos 4 melhores

BitTorrent: olho por olho

- (1) Alice “desafoga” Bob de forma “otimista”
- (2) Alice torna-se um dos quatro maiores provedores de Bob; Bob retribui
- (3) Bob torna-se um dos quatro melhores provedores de Alice



maior taxa de upload: encontra melhores parceiros, obtém arquivo mais rápido!

Camada de aplicação: visão geral

- princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- o Domain Name System DNS
- aplicações P2P
- transmissão de vídeo e redes de distribuição de conteúdo



Transmissão de vídeo e CDNs: contexto

- fluxo de tráfego de vídeo: grande consumidor de largura de banda da Internet
 - Netflix, YouTube, Amazon Prime: 80% do tráfego de ISP residencial (2020)
- *desafio:* escala - como alcançar ~1B usuários?
- *desafio:* heterogeneidade
 - diferentes usuários têm diferentes recursos (por exemplo, com fio versus móvel; alta largura de banda versus baixa largura de banda)
- *solução:* infraestrutura distribuída em nível de aplicação



Multimídia: vídeo

- vídeo: sequência de imagens exibidas a ritmo constante
 - Ex.: 24 imagens/segundo
- imagem digital: matriz de pixels
 - cada pixel representado por bits
- codificação: usar redundância *dentro de* e *entre* imagens para diminuir # bits usados para codificar imagem
 - espacial (dentro da imagem)
 - temporal (de uma imagem para a próxima)

exemplo de codificação espacial:
em vez de enviar N valores da mesma cor (todos roxos), enviar apenas dois valores: o valor da cor (roxo) e o número de repetições (N)



quadro *i*

exemplo de codificação temporal: em vez de enviar quadro completo em $i+1$, enviar apenas diferenças dele para o quadro i



quadro *i+1*

Multimídia: vídeo

- **CBR: (constant bit rate):** taxa de codificação de vídeo fixa
- **VBR: (variable bit rate):** taxa de codificação de video muda conforme ocorrem mudanças na quantidade informações espaciais e temporais codificadas
- **exemplos:**
 - MPEG 1 (CD-ROM) 1,5 Mbps
 - MPEG2 (DVD) 3-6 Mbps
 - MPEG4 (frequentemente usado na Internet, 64Kbps – 12 Mbps)

exemplo de codificação espacial:
em vez de enviar N valores da mesma cor (todos roxos), enviar apenas dois valores: o valor da cor (roxo) e o número de repetições (N)



quadro i

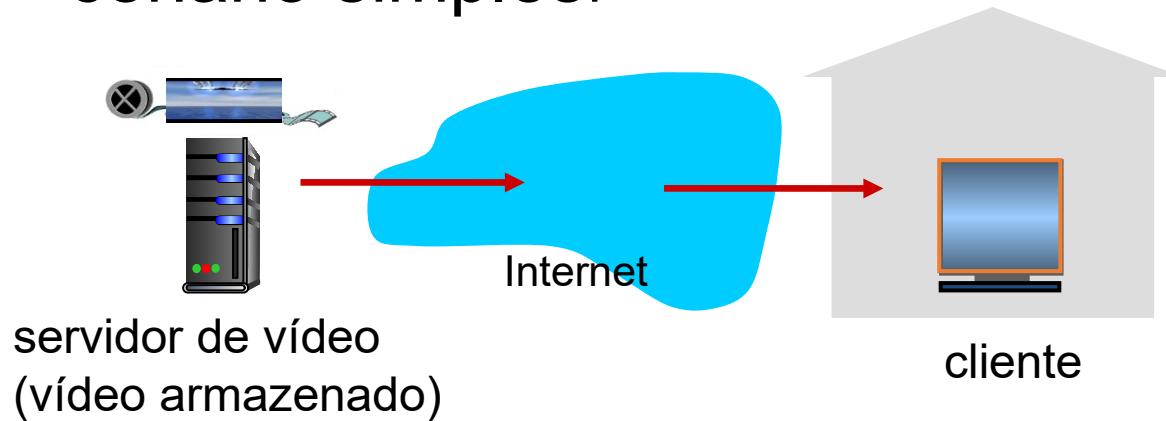
exemplo de codificação temporal: em vez de enviar quadro completo em $i+1$, enviar apenas diferenças dele para o quadro i



quadro $i+1$

Transmissão de vídeo armazenado

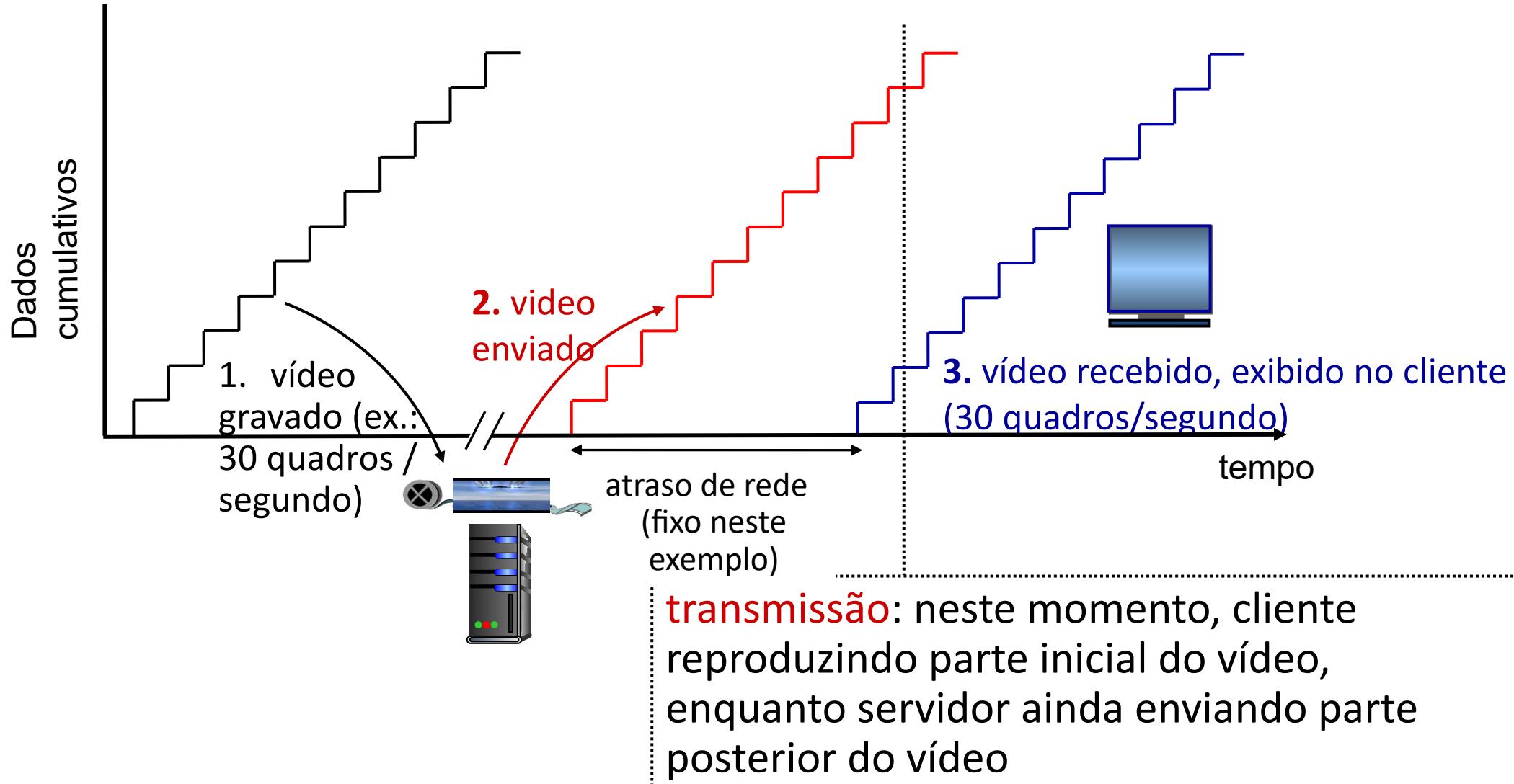
cenário simples:



Principais desafios:

- largura de banda de servidor para cliente vai *variar* ao longo do tempo, com a mudança dos níveis de congestionamento da rede (em casa, rede de acesso, núcleo de rede, servidor de vídeo)
- perda de pacote, atraso devido ao congestionamento vai atrasar a exibição, ou resultar em má qualidade de vídeo

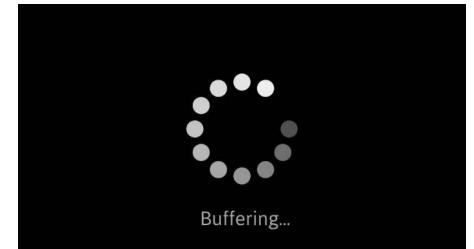
Transmissão de vídeo armazenado



Transmissão de vídeo armazenado: desafios

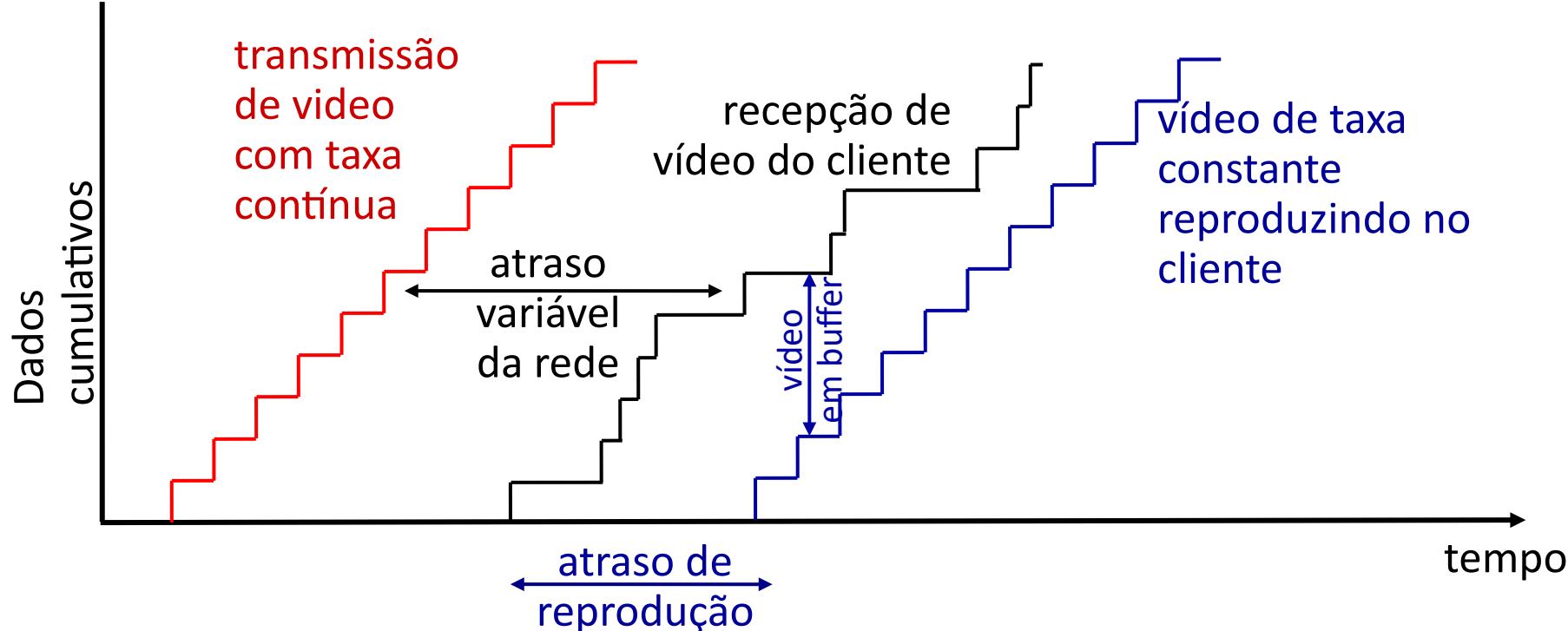
- **restrição de reprodução contínua:** durante a reprodução de vídeo do cliente, o tempo de reprodução deve corresponder ao tempo original

- ... mas **atrasos na rede são variáveis** (jitter), por isso vai precisar **buffer** do lado do cliente para corresponder à restrição de reprodução contínua



- outros desafios:
 - interatividade do cliente: pausar, avançar, retroceder, pular no vídeo
 - pacotes de vídeo podem ser perdidos, retransmitidos

Transmissão de vídeo armazenado: buffer de reprodução



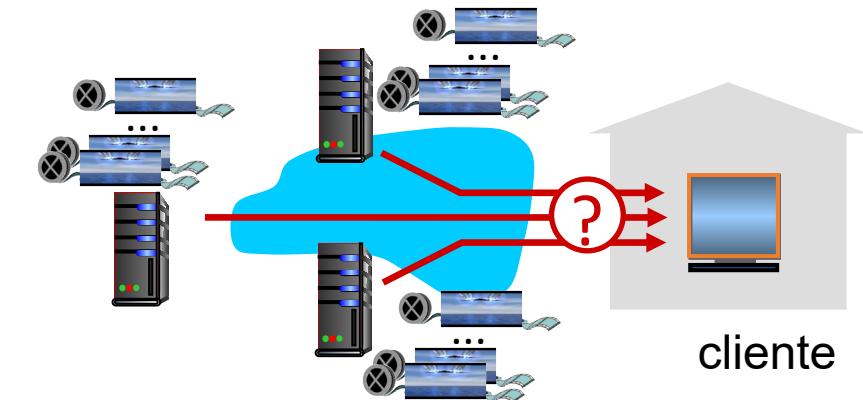
- *buffer do lado do cliente e atraso de reprodução:* compensa o atraso adicionado pela rede e a variação de atraso (*jitter*)

Transmissão multímidia: DASH

Dynamic, Adaptive
Streaming over HTTP

servidor:

- divide arquivo de vídeo em vários pedaços
- cada pedaço codificado em múltiplas taxas diferentes
- codificações de taxas diferentes armazenadas em arquivos diferentes
- arquivos replicados em vários nós CDN
- *arquivo de manifesto*: fornece URLs para diferentes pedaços



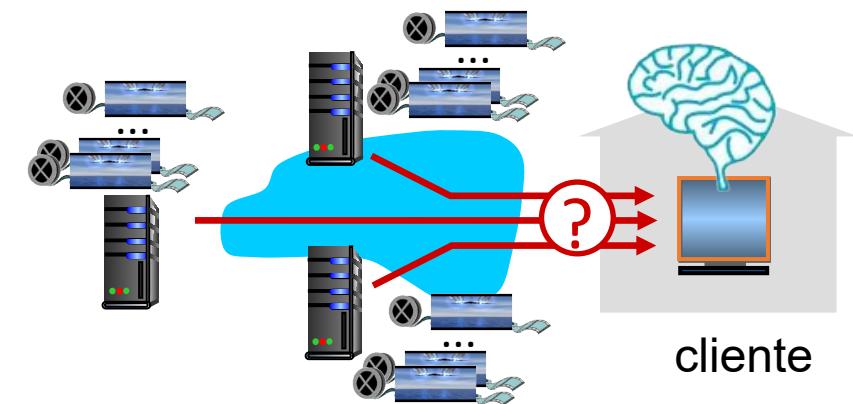
cliente:

- periodicamente estima largura de banda de servidor para cliente
- consulta manifesto, solicita um pedaço de cada vez
 - escolhe taxa máxima de codificação sustentável dada a largura de banda atual
 - pode escolher diferentes taxas de codificação em diferentes pontos no tempo (dependendo da largura de banda disponível no momento), e de diferentes servidores

Transmissão multimídia: DASH

- “*inteligência*” no cliente: cliente determina:

- *quando* solicitar pedaço (para que não ocorra estouro ou esvaziamento de buffer)
- *qual taxa de codificação* solicitar (maior qualidade quando mais largura de banda está disponível)
- *de onde* solicitar o pedaço (pode solicitar da URL do servidor que está “perto” do cliente ou tem alta largura de banda disponível)



Transmissão de vídeo = codificação + DASH + buffer de reprodução

Redes de Distribuição de Conteúdo- Content distribution networks (CDNs)

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

- **opção 1:** único, grande “mega-servidor”
 - único ponto de falha
 - ponto de congestionamento de rede
 - longo (e possivelmente congestionado) caminho para clientes distantes

.... muito simples: esta solução *não escala*

Redes de Distribuição de Conteúdo- Content distribution networks (CDNs)

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

- **opção 2:** armazenar/servir várias cópias de vídeos em vários sites geograficamente distribuídos (*CDN*)
 - *enter deep:* empurrar servidores CDN profundamente em muitas redes de acesso
 - perto dos usuários
 - Akamai: 240.000 servidores implantados em > 120 países (2015)
 - *bring home:* número menor (dezenas) de clusters maiores em POPs perto de redes de acesso
 - usado pela Limelight



Akamai hoje:



The Akamai Edge Today

360K
servers

100+
million hits
per second

7+
trillion
deliveries
per day

175+
terabits per
second
(250+ peak)

4,200+
locations

1,350+
networks

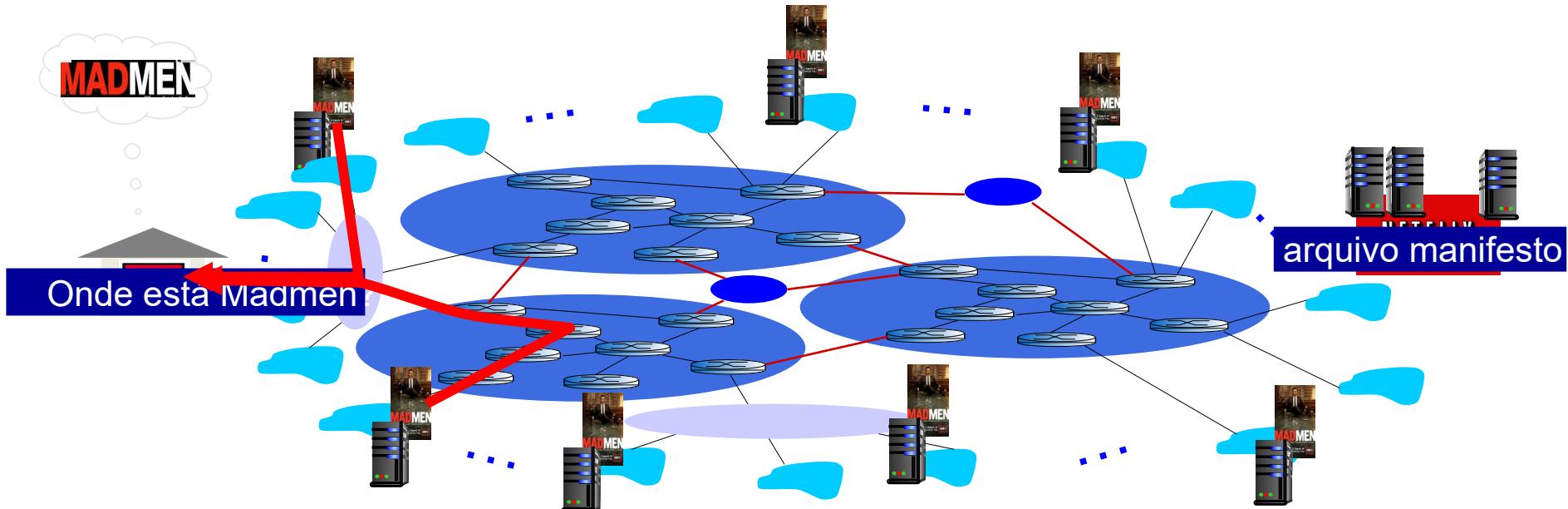
840+
cities

135
countries

Fonte: <https://networkingchannel.eu/living-on-the-edge-for-a-quarter-century-an-akamai-retrospective-downloads/>

Como a Netflix funciona?

- Netflix: armazena cópias de conteúdo (por exemplo, MADMEN) nos nós (mundiais) de sua CDN chamada OpenConnect
- assinante solicita conteúdo, provedor de serviços retorna manifesto
 - usando manifesto, cliente recupera conteúdo na taxa mais alta suportada
 - pode escolher taxa ou cópia diferente se o caminho da rede estiver congestionado



Redes de Distribuição de Conteúdo- Content distribution networks (CDNs)



Comunicação de hospedeiro a hospedeiro da Internet como um serviço

Desafios do OTT: lidar com uma Internet congestionada a partir da “borda” que conteúdo colocar em qual nó da CDN?

- de qual nó da CDN recuperar conteúdo? A que taxa?