

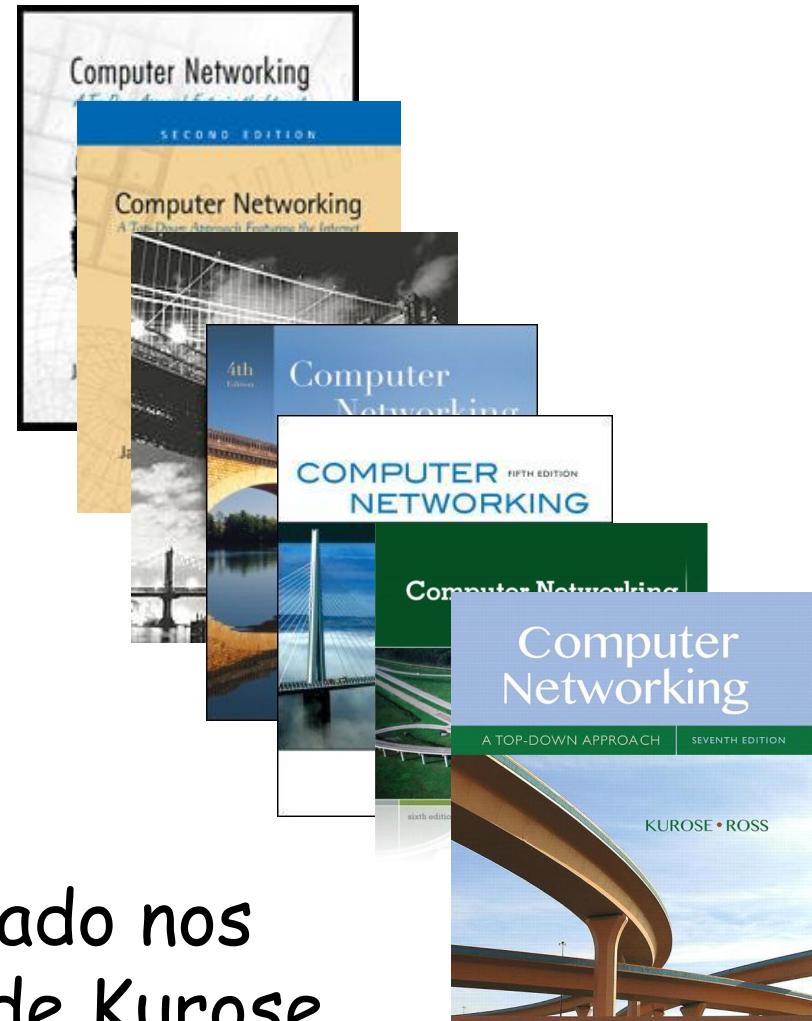
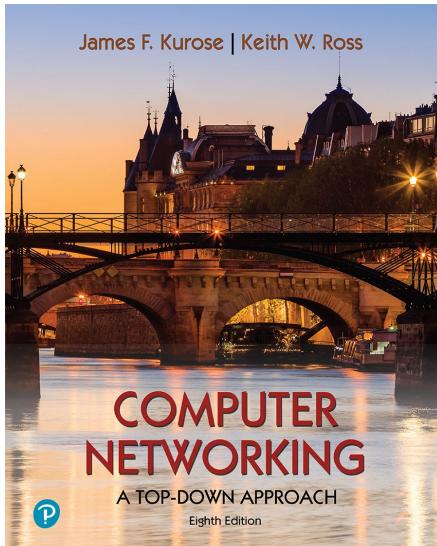


# Camada de Aplicações - Parte 01

## GEX105 - REDES DE COMPUTADORES

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.

# Livro-Texto:



Baseado nos  
slides de Kurose  
e Ross

# Camada de Aplicações

- Princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O Sistema de Nomes de Domínio (DNS)
- Aplicações P2P
- Transmissão de vídeo e redes de distribuição de conteúdo
- Programação de sockets com UDP e TCP

# Camada de Aplicações

Nossos objetivos:

- aspectos conceituais e de implementação de protocolos de camada de aplicação
  - modelos de serviço de camada de transporte
  - paradigma cliente-servidor
  - paradigma peer-to-peer

- Aprenda sobre protocolos examinando protocolos de camada de aplicação populares e infraestrutura:
  - HTTP
  - SMTP, IMAP
  - DNS
- Sistemas de streaming de vídeo, CDNs
- Programação de aplicações de rede
- API de sockets

# Algumas aplicações de rede

- Redes sociais
- Web
- Mensagens de texto
- e-mail
- Jogos de rede multiusuário
- Transmissão de vídeo armazenado (YouTube, Netflix)
- Compartilhamento de arquivos P2P
- voice over IP (e.g., Skype)
- real-time video conferencing (e.g., Zoom)
- Internet search
- remote login
- ...

*Q: Seu Favorito?*

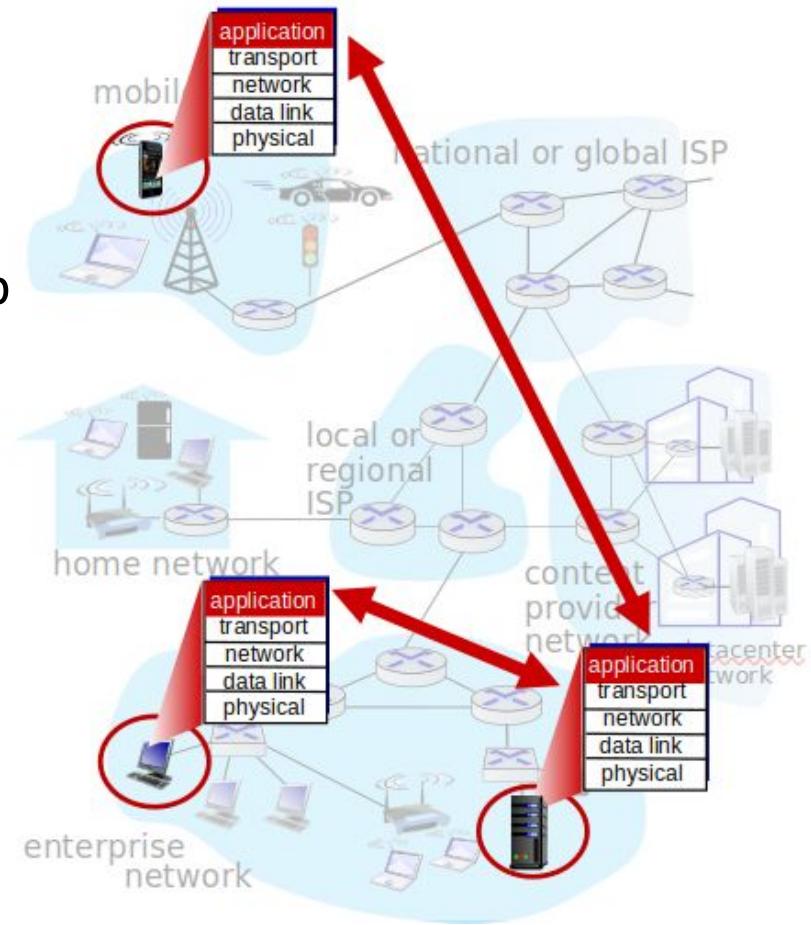
# Criando uma aplicação de rede

Escrever programas que:

- Executam em sistemas finais (diferentes)
- Comunicam-se pela rede
- Por exemplo, software de servidor web comunica-se com software de navegador

Não é necessário escrever software para dispositivos do núcleo da rede

- Dispositivos do núcleo da rede não executam aplicativos do usuário
- Aplicativos em sistemas finais permitem um desenvolvimento e propagação rápidos da aplicação



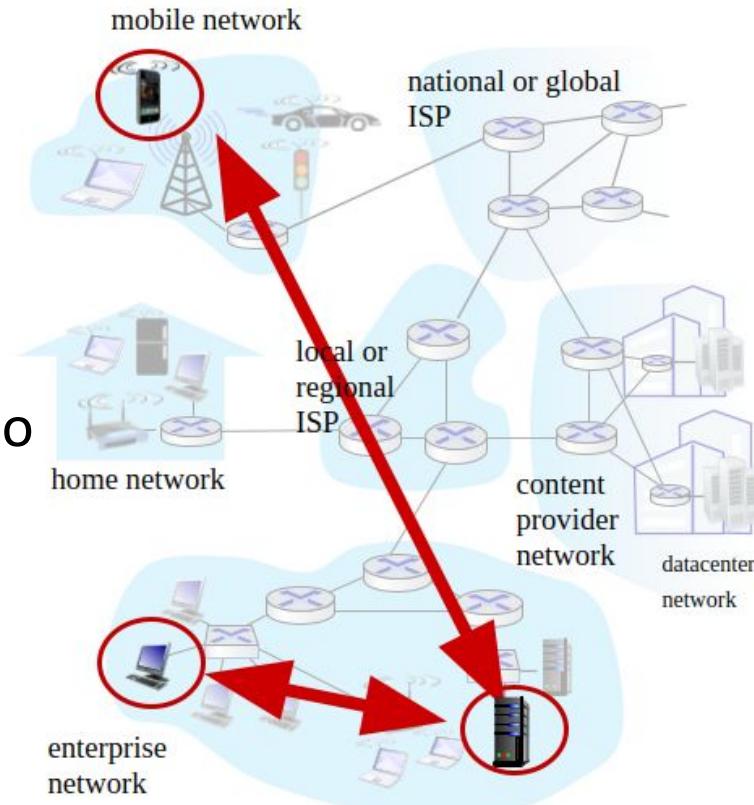
# Paradigma Cliente Servidor

## Servidor:

- Hospedeiro sempre ativo
- Endereço IP permanente
- Muitas vezes em centros de dados, para escalabilidade

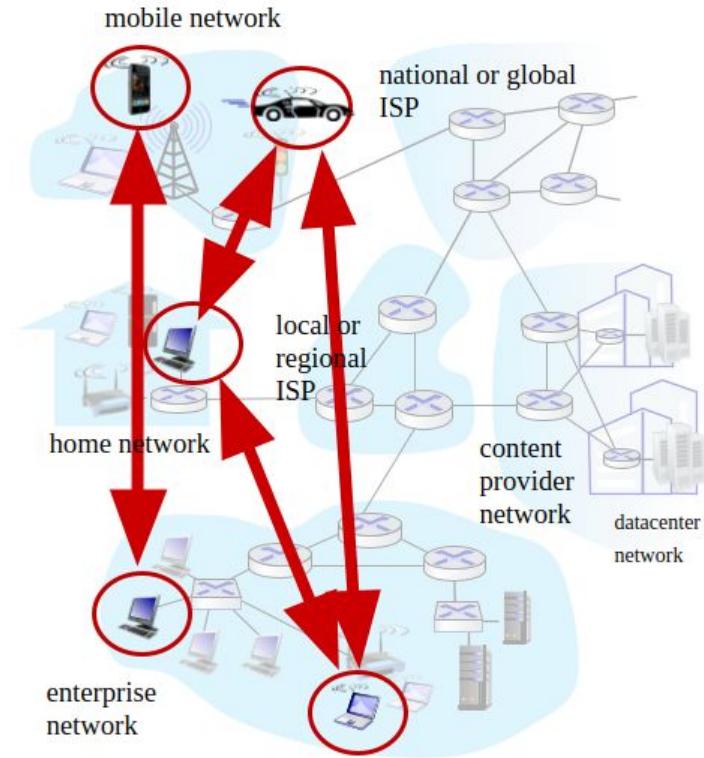
## Clientes:

- Entram em contato, comunicam-se com o servidor
- Podem estar conectados de forma intermitente
- Podem ter endereços IP dinâmicos
- Não se comunicam diretamente entre si
- Exemplos: HTTP, IMAP, FTP



# Arquitetura Peer-to-Peer - P2P

- Sem servidor sempre ativo
- Sistemas finais arbitrários se comunicam diretamente
- Pares solicitam serviço de outros pares e fornecem serviço em troca para outros pares
- Autoescalabilidade - novos pares trazem nova capacidade de serviço, assim como novas demandas de serviço
- Pares estão intermitentemente conectados e mudam de endereço IP
- Gerenciamento complexo
- Exemplo: Compartilhamento de arquivos P2P [BitTorrent]



# Comunicação de Processos

- **Processo:** programa em execução dentro de um hospedeiro
- Dentro do mesmo hospedeiro, dois processos se comunicam usando **comunicação entre processos** (definida pelo sistema operacional)
- Processos em hospedeiros diferentes comunicam-se trocando **mensagens**

clients, servers

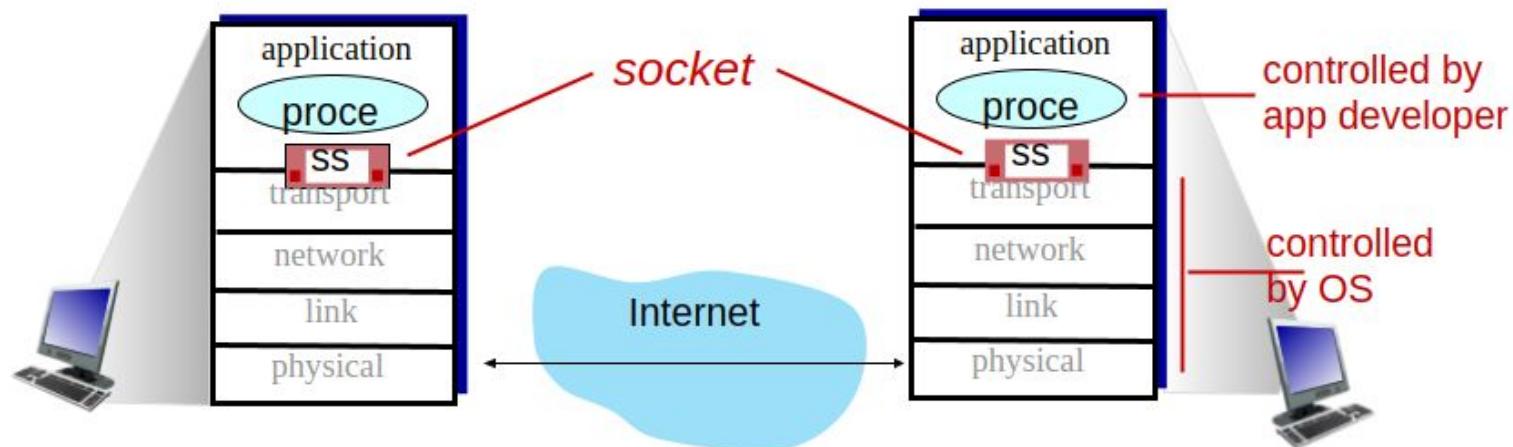
*client process:* process that initiates communication

*server process:* process that waits to be contacted

Nota: Aplicações com arquiteturas P2P têm processos clientes e processos servidores.

# Sockets

- O processo envia/recebe mensagens para/de seu socket.
- Um socket é análogo a uma porta.
  - O processo remetente empurra a mensagem para fora da porta.
  - O processo remetente depende da infraestrutura de transporte do outro lado da porta para entregar a mensagem ao socket do processo receptor.
  - Dois sockets estão envolvidos: um em cada lado.



# Endereçamento de processos

- Para receber mensagens, um processo deve ter um identificador.
- O dispositivo host tem um endereço IP único de 32 bits.

P: O endereço IP do host em que o processo é executado é suficiente para identificar o processo?

R: Não, muitos processos podem estar em execução no mesmo host.

- O **identificador** inclui tanto o **endereço IP** quanto os **números de porta** associados ao processo no host.
- Exemplos de números de porta:
  - Servidor HTTP: 80
  - Servidor de email: 25
- Para enviar uma mensagem HTTP para o servidor web `gaia.cs.umass.edu`:
  - Endereço IP: 128.119.245.12
  - Número da porta: 80

[https://pt.wikipedia.org/wiki/Lista\\_de\\_portas\\_dos\\_protocolos\\_TCP\\_e\\_UDP](https://pt.wikipedia.org/wiki/Lista_de_portas_dos_protocolos_TCP_e_UDP)

# Um protocolo de camada de aplicação define:

- Tipos de mensagens trocadas, por exemplo, solicitação, resposta.
- Sintaxe da mensagem: quais campos nas mensagens e como os campos são delineados.
- Semântica da mensagem: significado das informações nos campos.
- Regras para quando e como processos enviam e respondem a mensagens.

## Protocolos abertos:

- Definidos em RFCs (Request for Comments), todos têm acesso à definição do protocolo.
- Permitem interoperabilidade, por exemplo, HTTP, SMTP.

## Protocolos proprietários:

- Exemplo, Skype, Zoom

# De que serviços uma aplicação necessita?

## Integridade dos dados (sensibilidade a perdas)

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

## Temporização (sensibilidade a atrasos)

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

## Vazão (throughput)

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

## Segurança

- Criptografia, integridade dos dados, ...

# Requisitos de aplicações de rede selecionadas

application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

# Serviços de protocolos de transporte da Internet

## *Serviço TCP:*

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

## *Serviço UDP:*

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

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

# Aplicações e seus protocolos de transporte

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP [RFC 7230, 9110]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary HTTP [RFC 7230], DASH	TCP or UDP
streaming audio/video	proprietary HTTP [RFC 7230], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

# Tornando o TCP seguro

## TCP & UDP

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

## Secure Sockets Layer - SSL

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

## SSL está na camada de aplicação

- Aplicações usam bibliotecas SSL, que “falam” com o TCP

## API do socket SSL

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

# Camada de Aplicação 2: Roteiro

2.1 Princípios de aplicações de rede

**2.2 A Web e o HTTP**

2.3 Correio Eletrônico na Internet

2.4 DNS: o serviço de diretório da Internet

2.5 Aplicações P2P

2.6 Fluxos (*streams*) de vídeo e Redes de Distribuição de Conteúdo (CDNs)

2.7 Programação de *sockets* com UDP e TCP

# A Web e o HTTP

Primeiro, uma revisão...

- Páginas Web consistem de **objetos**, cada um pode ser armazenado em servidores diferentes
- um objeto pode ser um arquivo HTML, uma imagem JPEG, um applet Java, um arquivo de áudio,...
- Páginas Web consistem de um arquivo base HTML que inclui vários objetos referenciados. Cada objeto é endereçável por uma URL - Uniform Resource Locator

Exemplo de URL:

www.someschool.edu/someDept/pic.gif

nome do hospedeiro

nome do caminho

# Protocolo HTTP

**HTTP: *hypertext transfer protocol***

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



# Mais sobre o protocolo HTTP

Usa serviço de transporte  
TCP:

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

HTTP é “sem estado”

servidor não mantém informação sobre pedidos anteriores do cliente

Not  
Protocolos que mantêm  
“estado” são complexos!

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

# Conexões HTTP

## HTTP não persistente

1. Conexão TCP aberta
2. No máximo um objeto enviado pela conexão TCP
3. Conexão TCP fechada

Baixar múltiplos objetos requer o uso de múltiplas conexões

## HTTP persistente

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

# Exemplo de HTTP não persistente

Supomos que usuário digita a URL

www.algumaUniv.br/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.

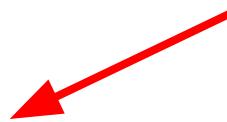
1b. servidor http no hospedeiro www.algumaUniv.br espera por conexão TCP na porta 80. "aceita" conexão, avisando ao cliente

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

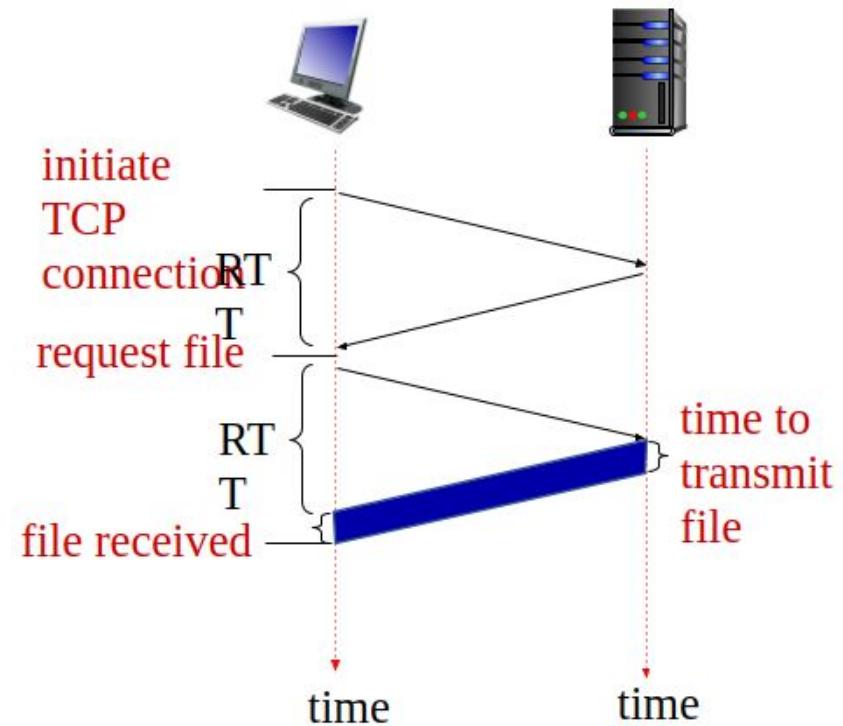
# Modelagem do tempo de resposta

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:

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



Non-persistent HTTP response time =  $2\text{RTT} + \text{tempo de transmissão do arquivo}$

# HTTP persistente (HTTP 1.1)

## Problemas com o HTTP não persistente:

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

## HTTP persistente (HTTP1.1)

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

# Mensagem de requisição HTTP

Dois tipos de mensagem HTTP: *requisição, resposta*  
*mensagem de requisição HTTP:*

ASCII (formato legível por pessoas)

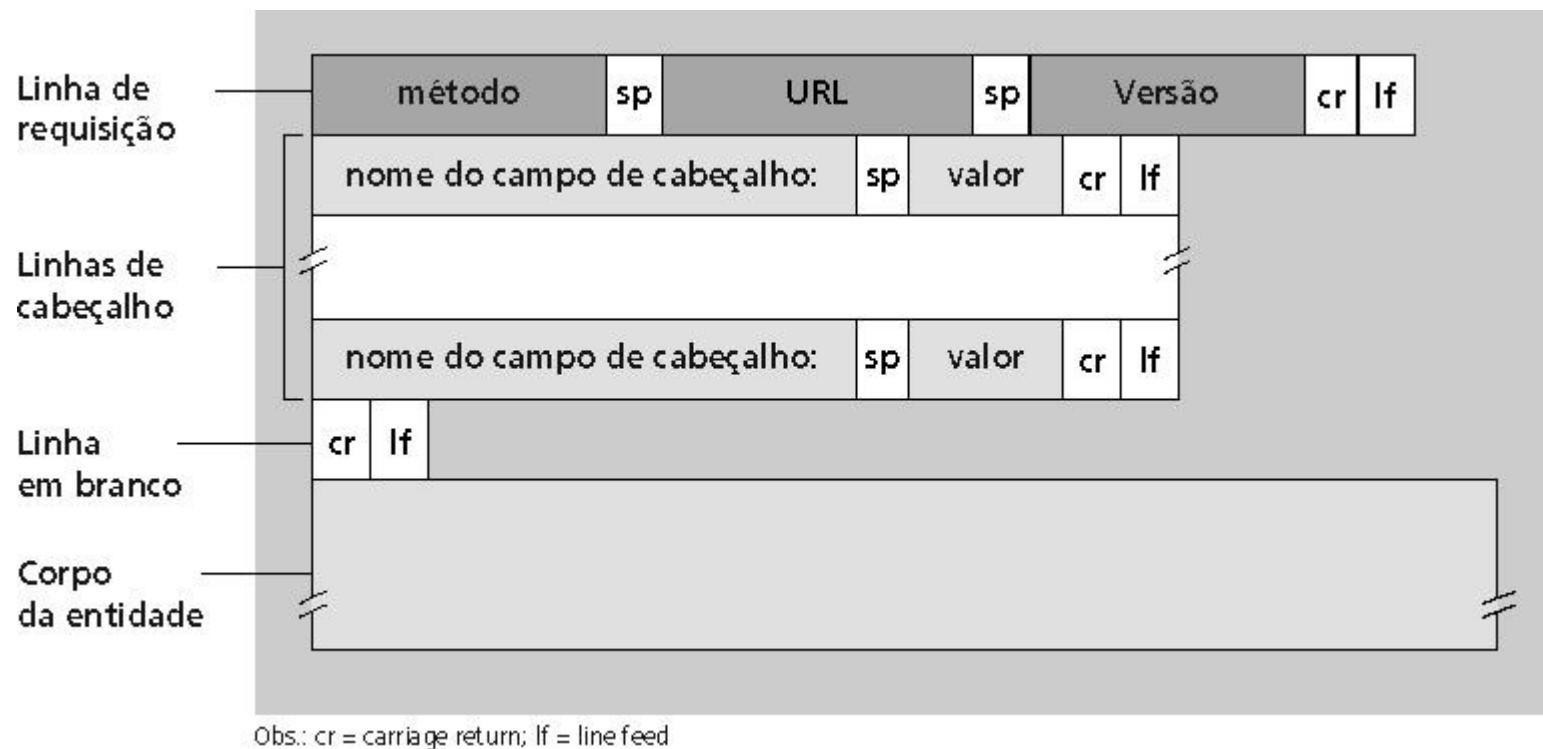
linha da requisição  
(comandos GET,  
POST, HEAD)

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



# Enviando conteúdo de formulário

## Método POST :

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

## Método URL:

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

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

# Tipos de métodos

## HTTP/1.0

- GET
- POST
- HEAD

Pede para o servidor não enviar o objeto requerido junto com a resposta

## HTTP/1.1

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

# Mensagem de resposta HTTP

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**

sucesso, objeto pedido segue mais adiante nesta mensagem

## **301 Moved Permanently**

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

## **400 Bad Request**

mensagem de pedido não entendida pelo servidor

## **404 Not Found**

documento pedido não se encontra neste servidor

## **505 HTTP Version Not Supported**

versão de http do pedido não usada por este servidor

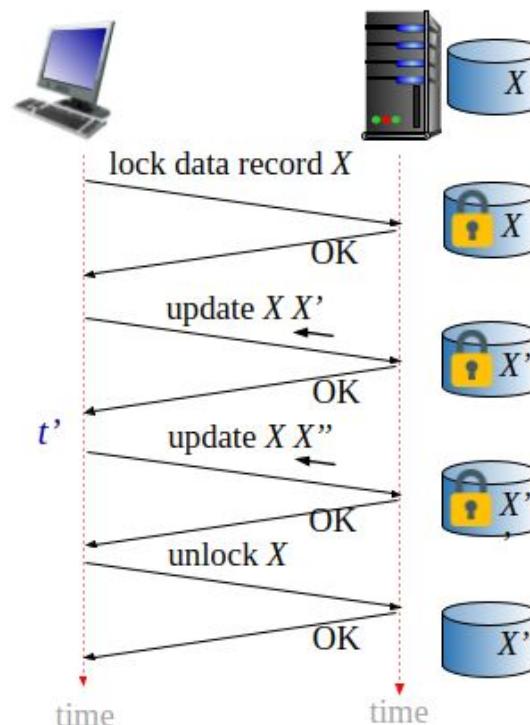
# Cookies: manutenção do “estado” da conexão

Lembre-se: A interação HTTP GET/resposta é sem estado.

Não há noção de trocas de mensagens HTTP de vários passos para concluir uma “transação” na Web.

- Não há necessidade de o cliente/servidor rastrear o “estado” da troca de vários passos.
- Todas as solicitações HTTP são independentes umas das outras.
- Não há necessidade de o cliente/servidor “recuperar” de uma transação parcialmente concluída, mas nunca completamente concluída.

a stateful protocol: client makes two changes to X, or none at all



Q: what happens if network connection or client crashes at  $t'$ ?

# Cookies: manutenção do “estado” da conexão

Sites da web e navegadores de clientes usam cookies para manter algum estado entre transações.

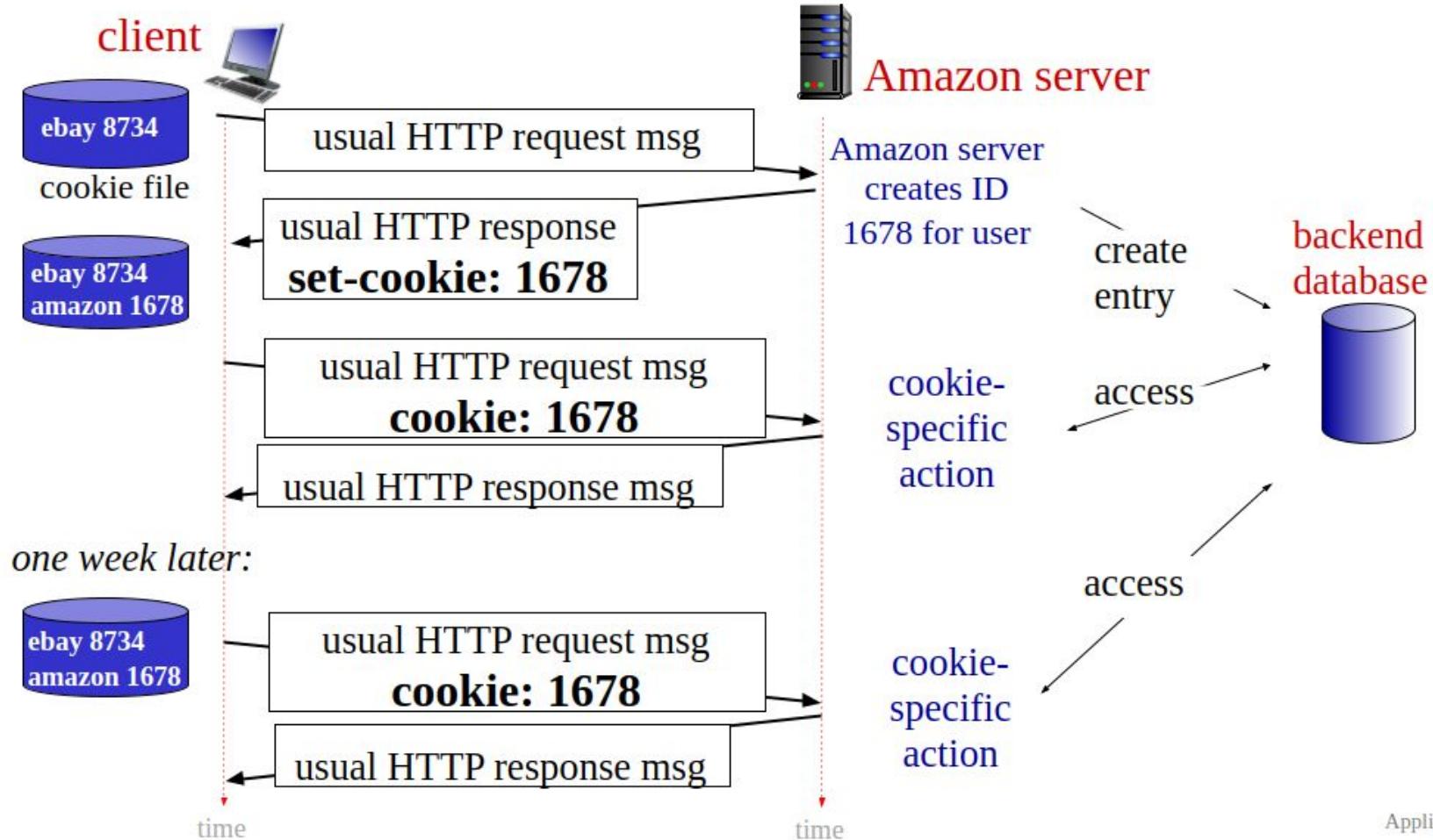
Quatro componentes:

1. Linha de cabeçalho de cookie da mensagem de resposta HTTP
2. Linha de cabeçalho de cookie na próxima mensagem de solicitação HTTP
3. Arquivo de cookie mantido no host do usuário, gerenciado pelo navegador do usuário
4. Banco de dados backend no site da web

Exemplo:

- Susan usa o navegador em seu laptop, visita um site de comércio eletrônico específico pela primeira vez.
- Quando a solicitação HTTP inicial chega ao site, o site cria:
  - um ID único (também conhecido como "cookie")
  - uma entrada no banco de dados de backend para o ID
- solicitações HTTP subsequentes de Susan para este site conterão o valor do ID do cookie, permitindo que o site "identifique" Susan.

# Cookies: manutenção do “estado” da conexão



# Cookies (continuação)

## O que os cookies podem obter:

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

**nota**

## Cookies e privacidade:

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

# *Exercício de Discussão: O Uso de Cookies na Web*

Leia:

- <https://goadopt.io/blog/cookies-e-lgpd/>
- <https://rockcontent.com/br/blog/google-adia-third-party-cookies/>

Após a leitura da notícia, vamos discutir sobre o uso de cookies na web.  
Considere os seguintes pontos para discussão:

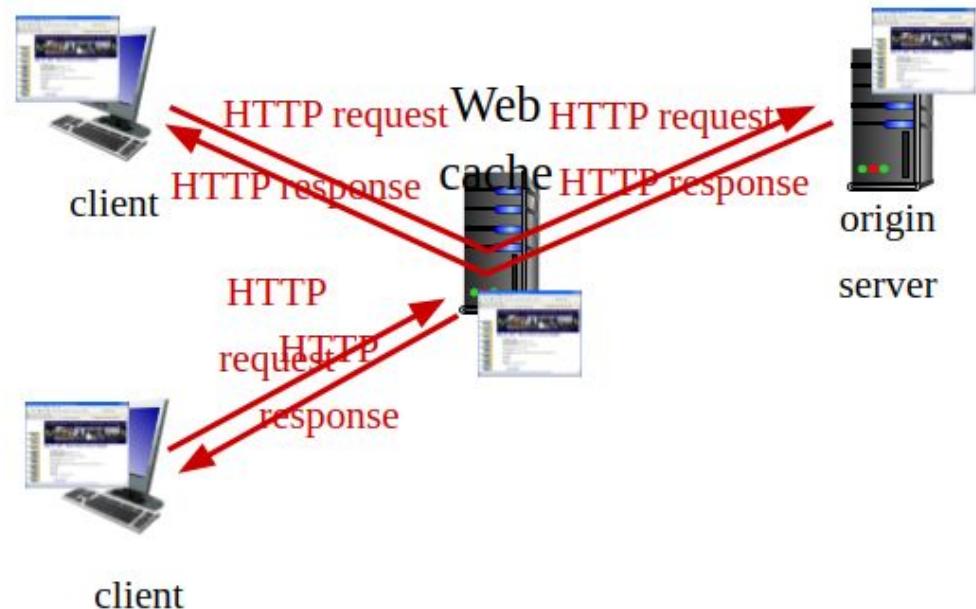
Qual é a função dos cookies na web?

1. Quais são os diferentes tipos de cookies mencionados na notícia?  
Explique-os.
2. Quais são os possíveis impactos da decisão do Google de adiar a eliminação dos cookies de terceiros até 2023?
3. De que maneira os cookies podem afetar a privacidade dos usuários da web?
4. Quais são as vantagens e desvantagens do uso de cookies para os usuários e para as empresas

# Web caches

**Meta:** atender pedido do cliente sem envolver servidor de origem

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



# Mais sobre Caches Web

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

O servidor informa ao cache sobre o cache permitido do objeto no cabeçalho de resposta:

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

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

## Para que fazer cache Web?

- Redução do tempo de resposta para os pedidos do cliente
- O cache está mais próximo do cliente
- Redução do tráfego no canal de acesso de uma instituição/Provedor
- Permite que provedores de conteúdo "fracos" entreguem conteúdo de forma mais eficaz

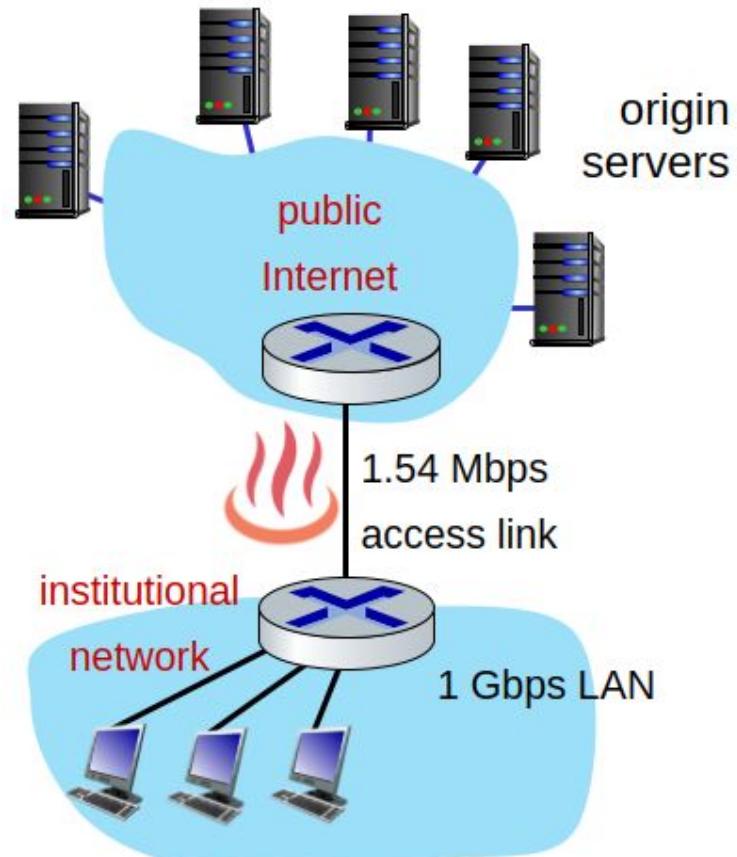
# Exemplo de cache (1)

## Hipóteses

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

## Consequências

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



# Opção 1: Adquirir um Link de acesso mais veloz

## Solução em potencial

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

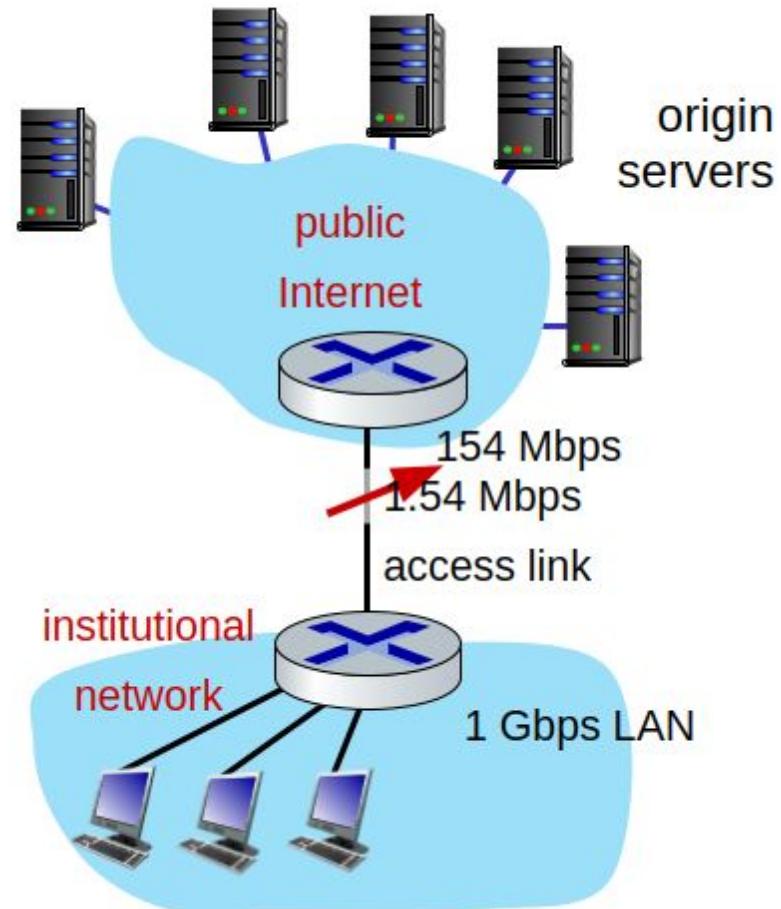
## Consequências

Utilização da LAN = 0,15%

Utilização do canal de acesso = **9,9%**

Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + msegs + microsssegundos

Frequentemente este é uma ampliação cara



# Opção 2: Instalação de um cache

## Instale uma cache

Assuma que a taxa de acerto seja de 0,4

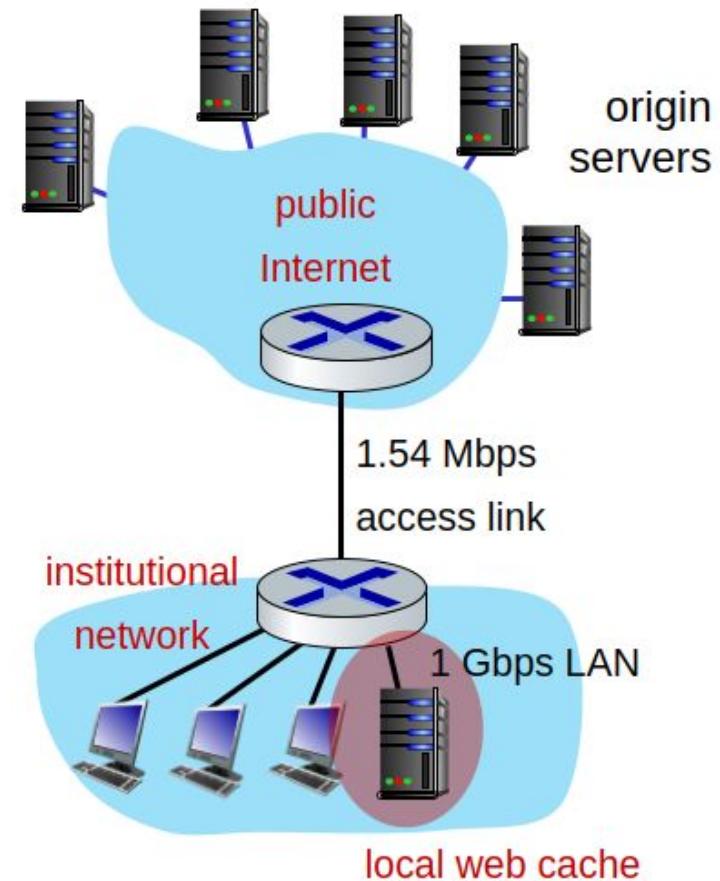
## Consequências

40% dos pedidos serão atendidos quase que imediatamente

60% dos pedidos serão servidos pelos servidores de origem

Utilização do canal de acesso é reduzido para 60%, resultando em atrasos desprezíveis (ex. 10 mseg)

Atraso total = atraso da Internet + atraso de acesso + atraso na LAN  
=  $0,6 \cdot 2 \text{ seg} + 0,6 \cdot 0,01 \text{ segs} +$   
mseg < 1,3 segs



# CDN - Content Delivery Network

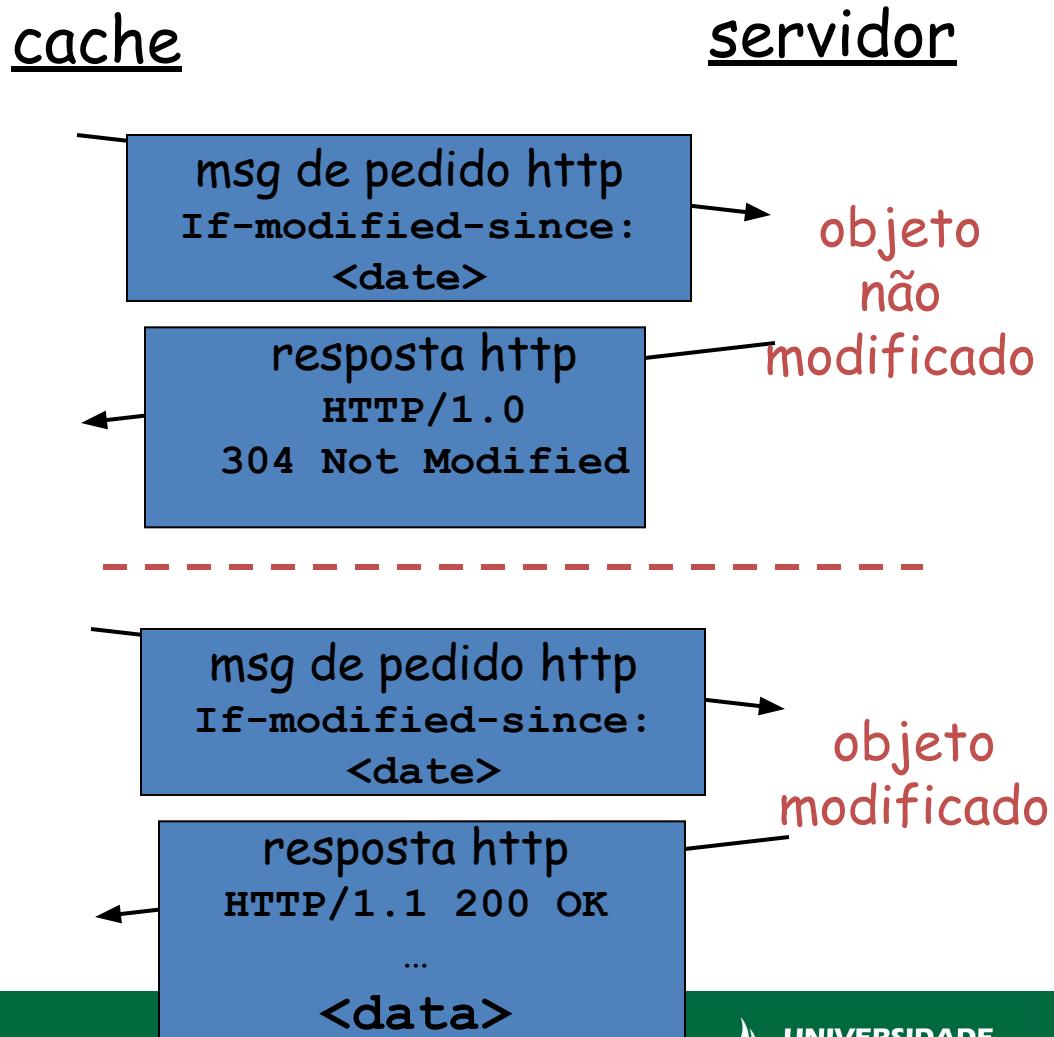
- Akamai (AANP)
- Apple
- Facebook (FNA)
- Google (GGC)
- Globo (GCA)
- Edgecast Verizon Digital Media (EC)
- Netflix (OCA)
- Azion (ANA)
- SourceForge.Net
- Microsoft
- CloudFlare

[https://wiki.brasilpeeringforum.org/w/CDN\\_Peering\\_e\\_PNI\\_-\\_Brasil](https://wiki.brasilpeeringforum.org/w/CDN_Peering_e_PNI_-_Brasil)

<https://opencdn.nic.br/> - [https://www.youtube.com/watch?v=QDpZM3vO\\_nM](https://www.youtube.com/watch?v=QDpZM3vO_nM)

# GET condicional

- **Meta:** não enviar objeto se cliente já tem (no cache) versão atual
  - Sem atraso para transmissão do objeto
  - Diminui a utilização do enlace
- cache: especifica data da cópia no cache no pedido HTTP  
**If-modified-since:**  
    `<date>`
- servidor: resposta não contém objeto se cópia no cache for atual:  
`HTTP/1.0 304 Not Modified`



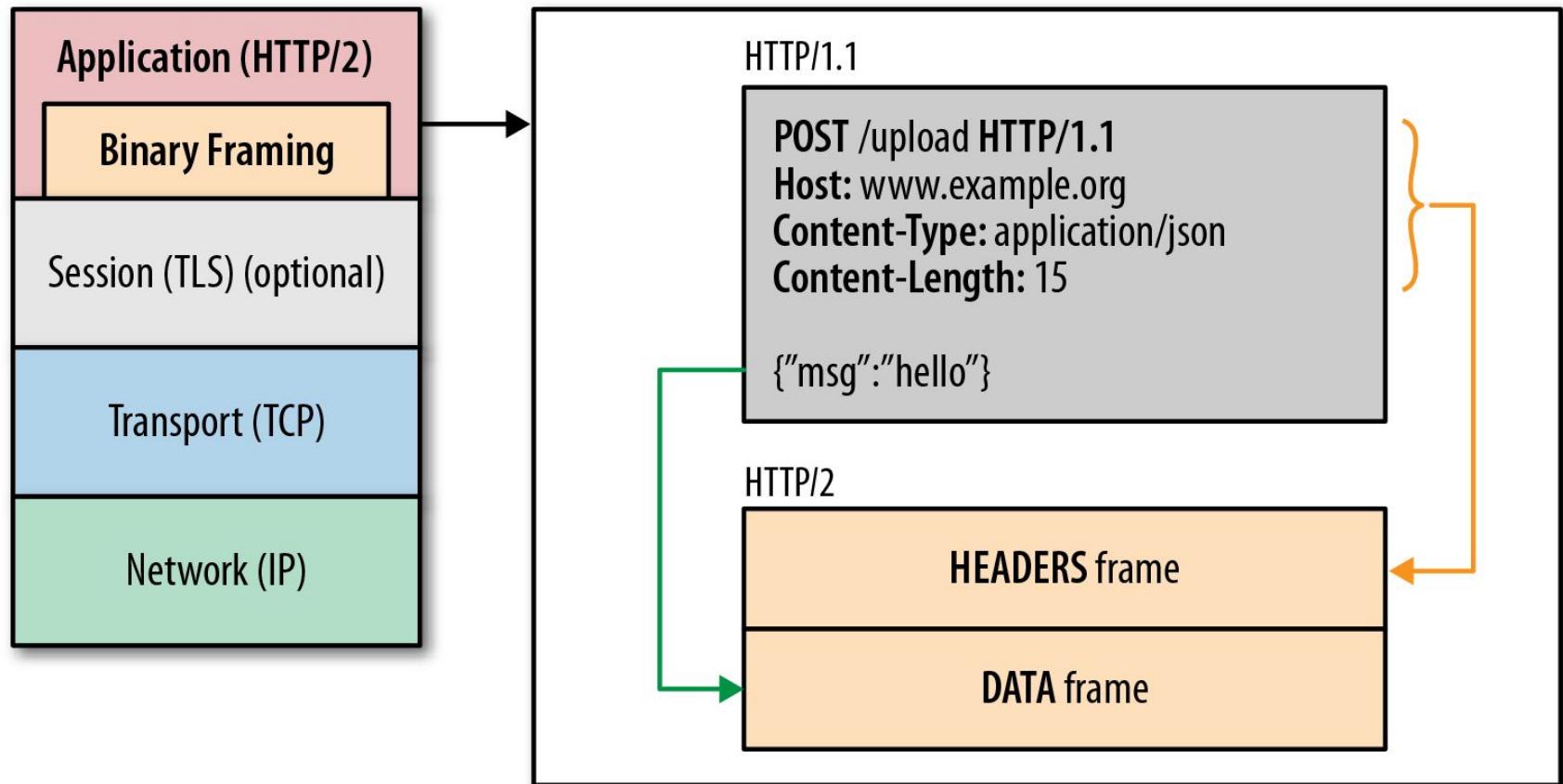
# HTTP/2

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

# HTTP/2: Diferenças do HTTP 1.1

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

# HTTP/2: Transporte Binário



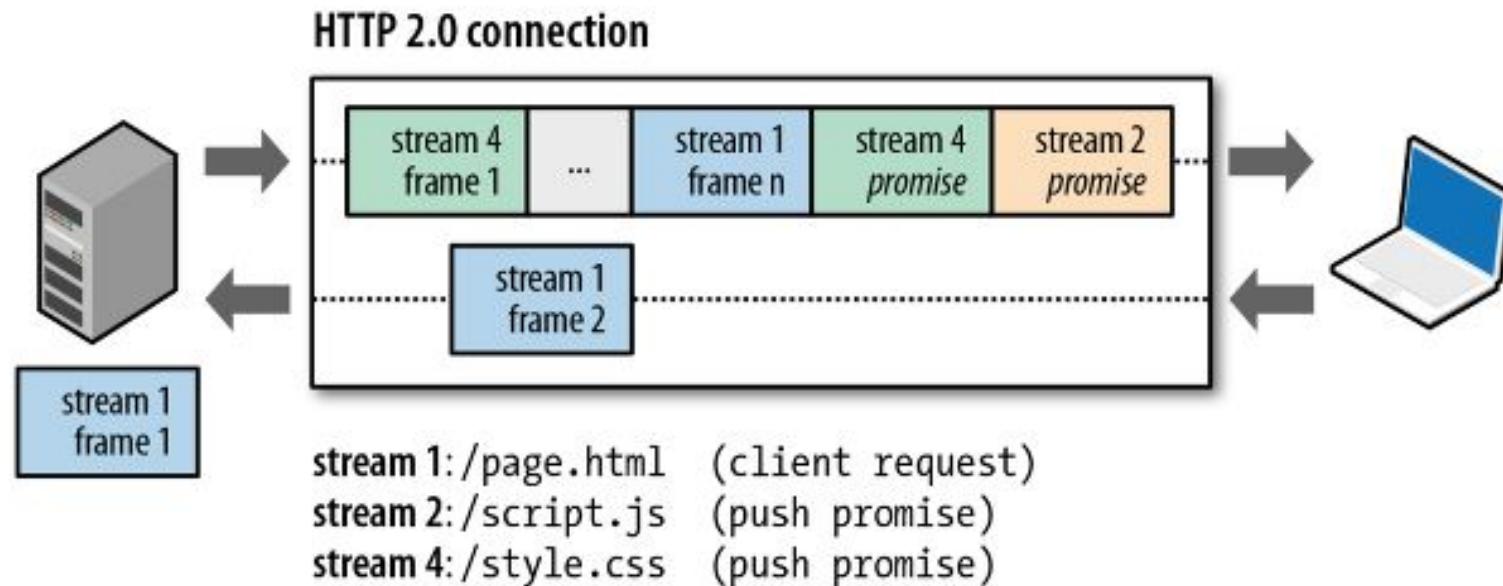
## HTTP/2: Quadros



Tipos:

HEADERS, DATA, PRIORITY, RST\_STREAM, SETTINGS,  
PUSH\_PROMISE, PING, GOAWAY, WINDOW\_UPDATE,  
CONTINUATION

# HTTP/2: Multiplexação



# HTTP/2 para HTTP/3

HTTP/2 sobre uma única conexão TCP significa:

- recuperação 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
- [https://youtu.be/VONSx\\_ftkz8?feature=shared&t=241](https://youtu.be/VONSx_ftkz8?feature=shared&t=241)

# Camada de Aplicações

- Princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O Sistema de Nomes de Domínio (DNS)
- Aplicações P2P
- Transmissão de vídeo e redes de distribuição de conteúdo
- Programação de sockets com UDP e TCP

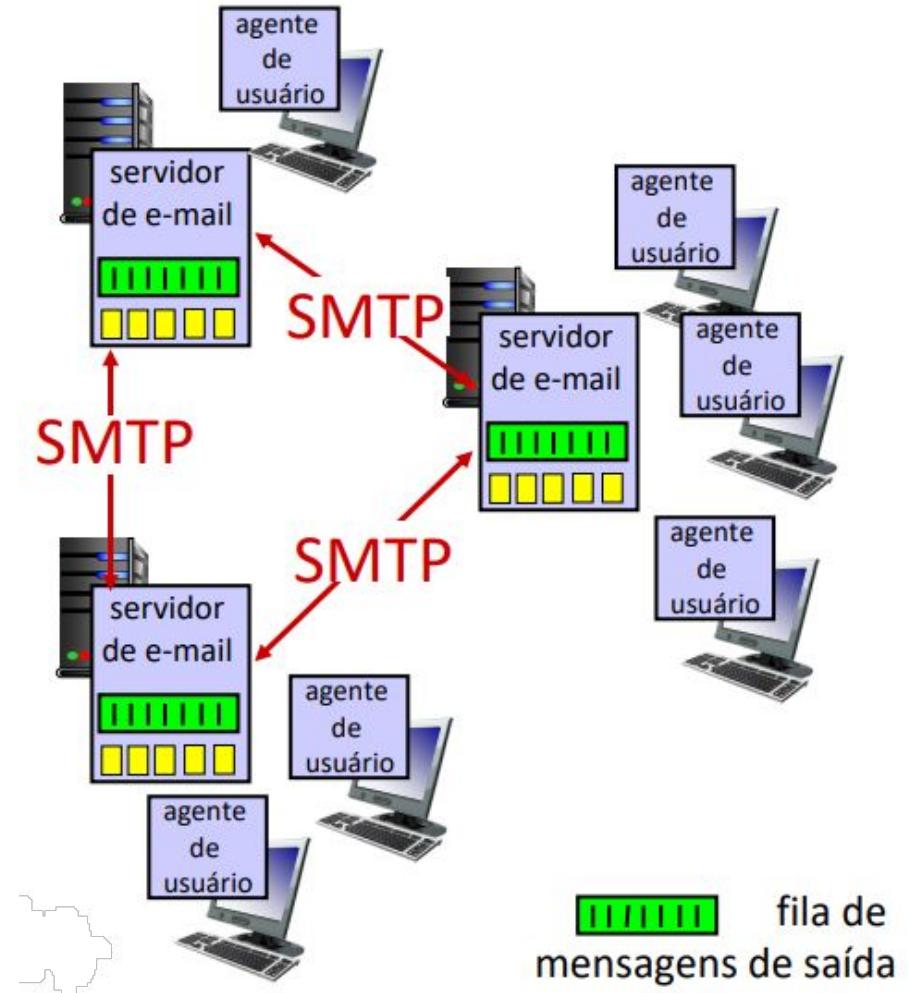
# Correio Eletrônico/E-mail

Três grandes componentes:

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

## Agente de Usuário

- também chamado “leitor de e-mail”
- compor, editar, ler mensagens de correio
- ex., Outlook, Thunderbird, cliente de mail do iPhone
- mensagens de saída e chegando são armazenadas no servidor



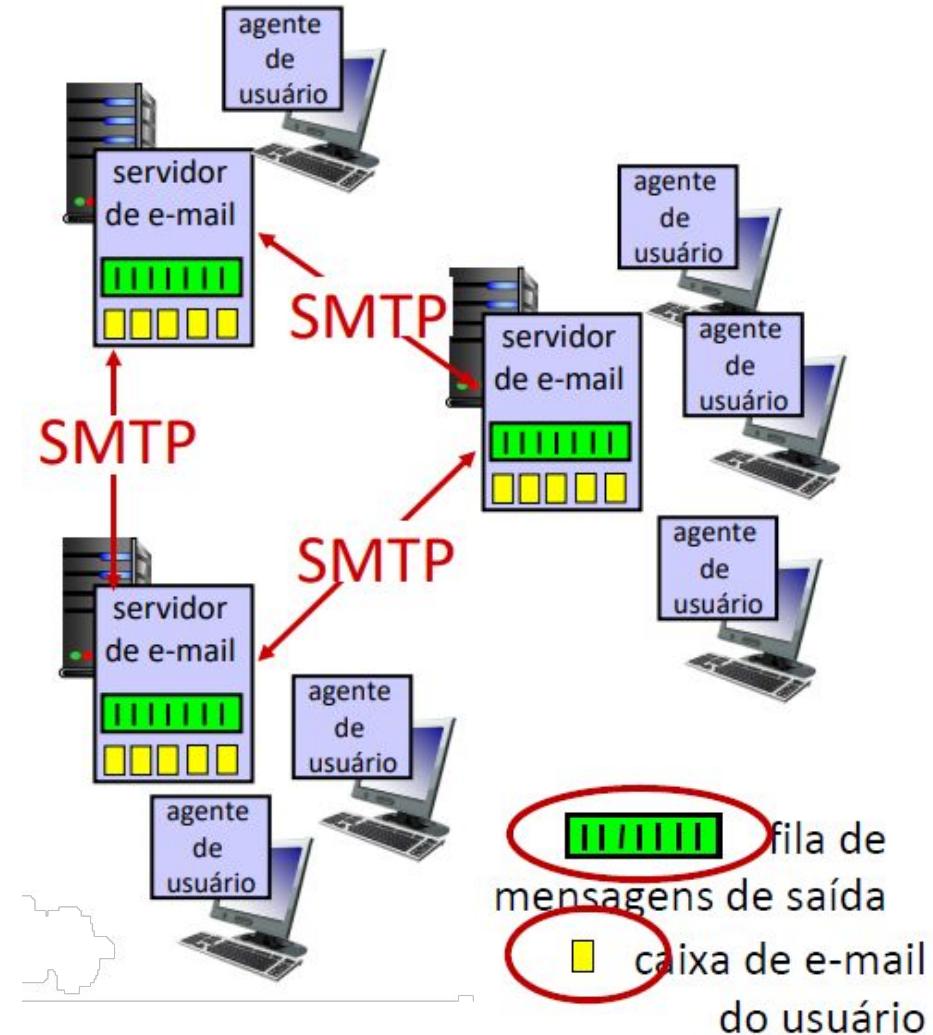
# Correio Eletrônico/E-mail

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)

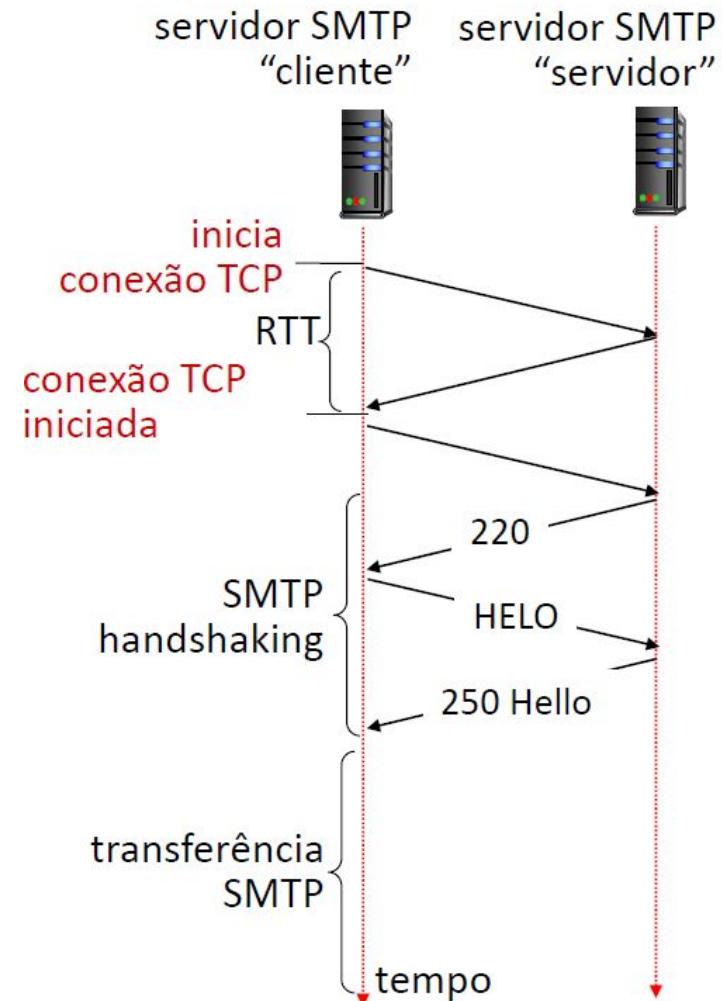
**protocolo SMTP** entre servidores de e-mail para enviar mensagens de e-mail

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



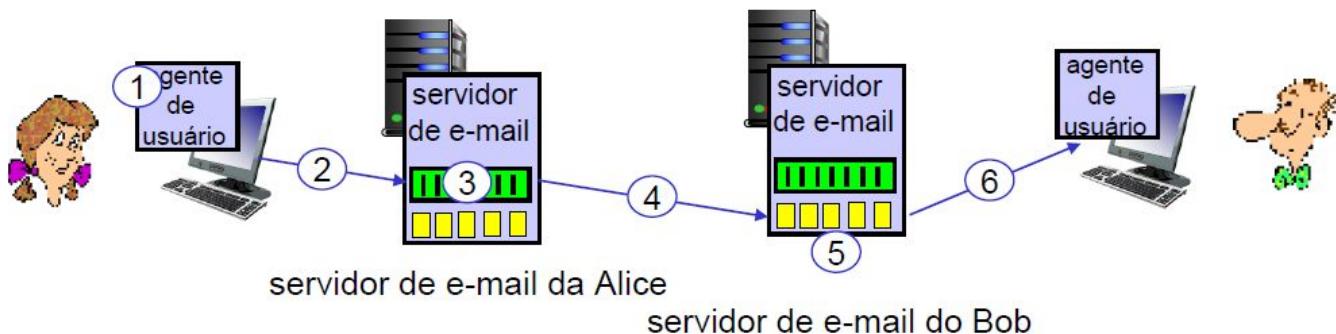
# SMTP RFC (5321)

- 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](mailto: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



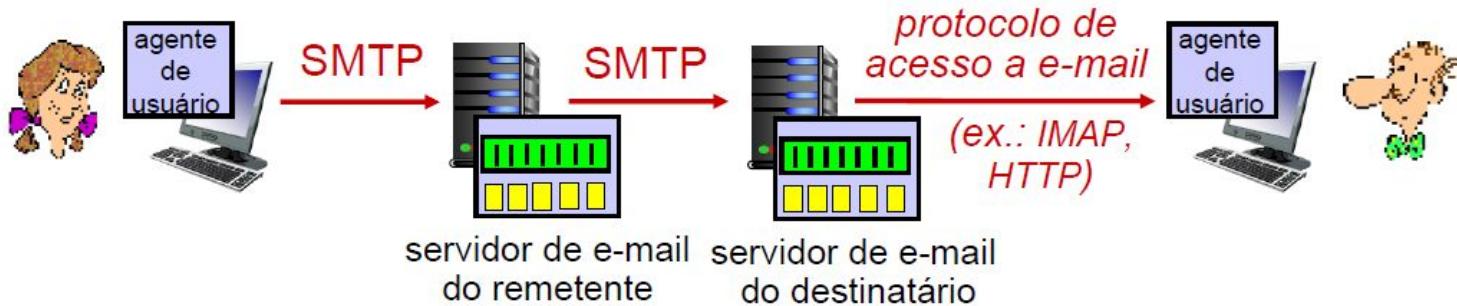
# Interação SMTP típica

```
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
```

# 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 STMP (para enviar), IMAP (ou POP) para recuperar mensagens de email

# Gerência da Porta 25

Configure a  
porta de envio  
de suas mensagens para

**587!**

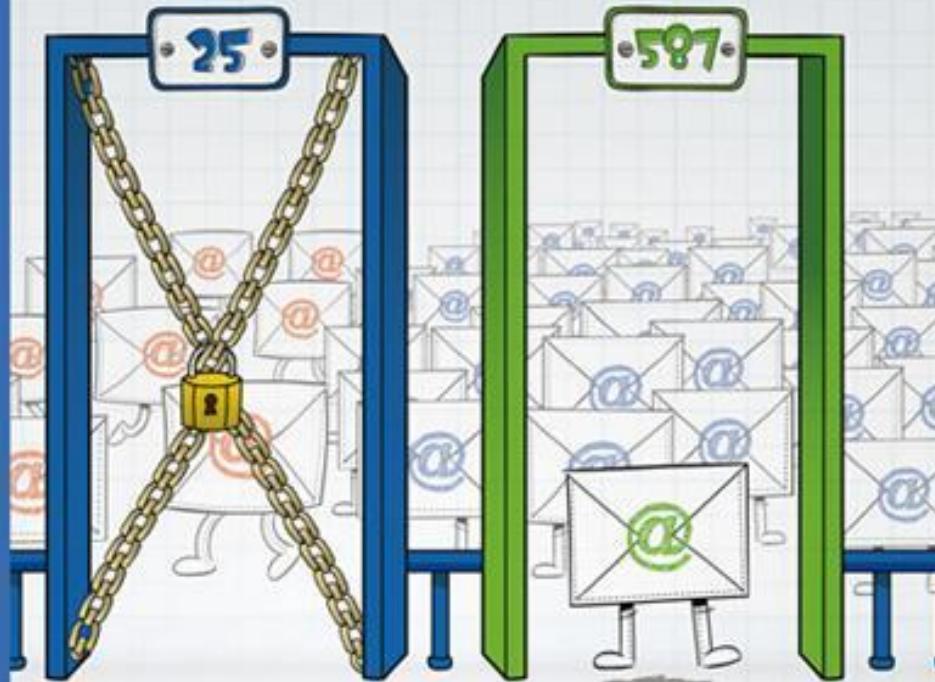
Com a Gerência da Porta 25, o Brasil vai reduzir  
o volume de spams enviados em nosso país.

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!



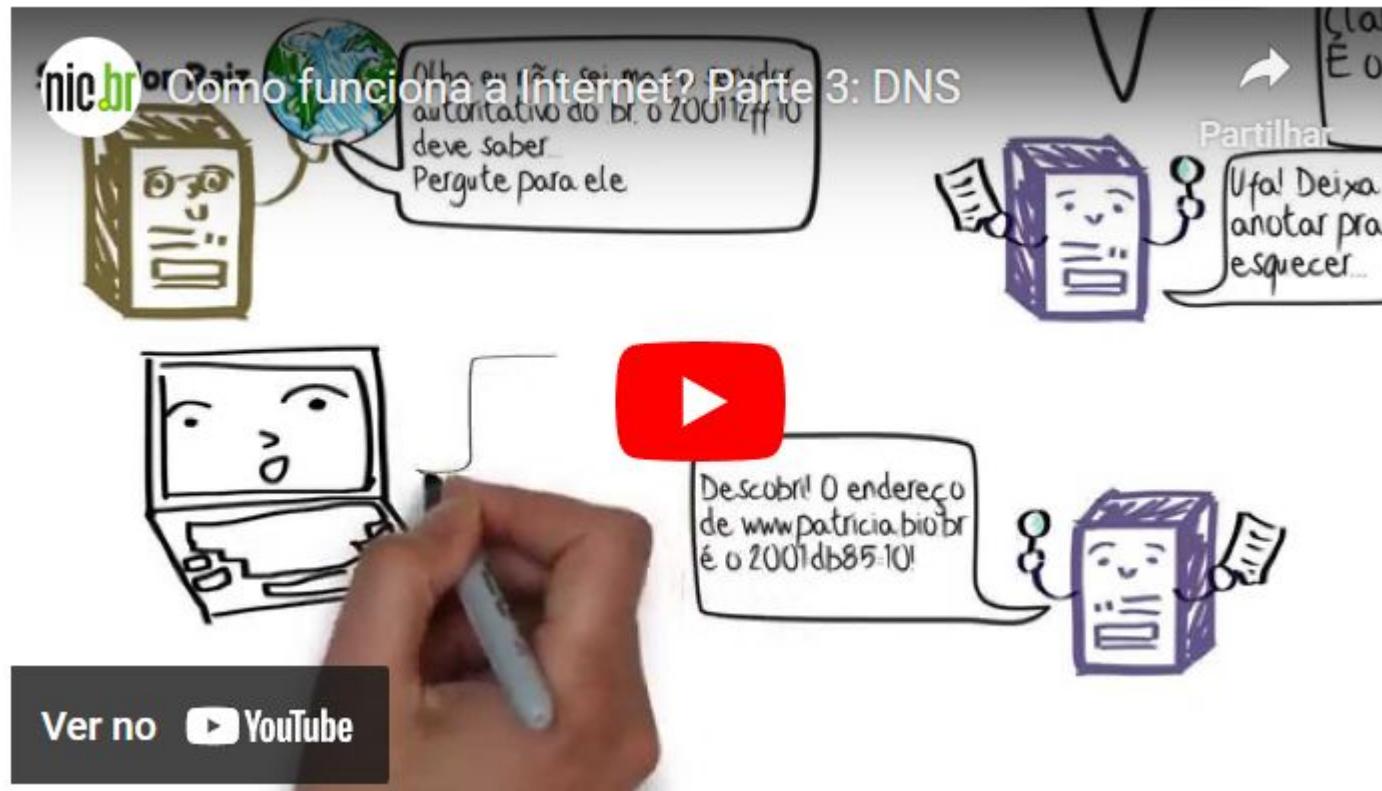
<http://antispam.br/>

# Camada de Aplicações

- Princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- **O Sistema de Nomes de Domínio (DNS)**
- Aplicações P2P
- Transmissão de vídeo e redes de distribuição de conteúdo

# DNS: Domain Name System

<https://www.nic.br/videos/ver/como-funciona-a-internet-parte-3-dns/>



# 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, www.uffs.edu.br - 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:**

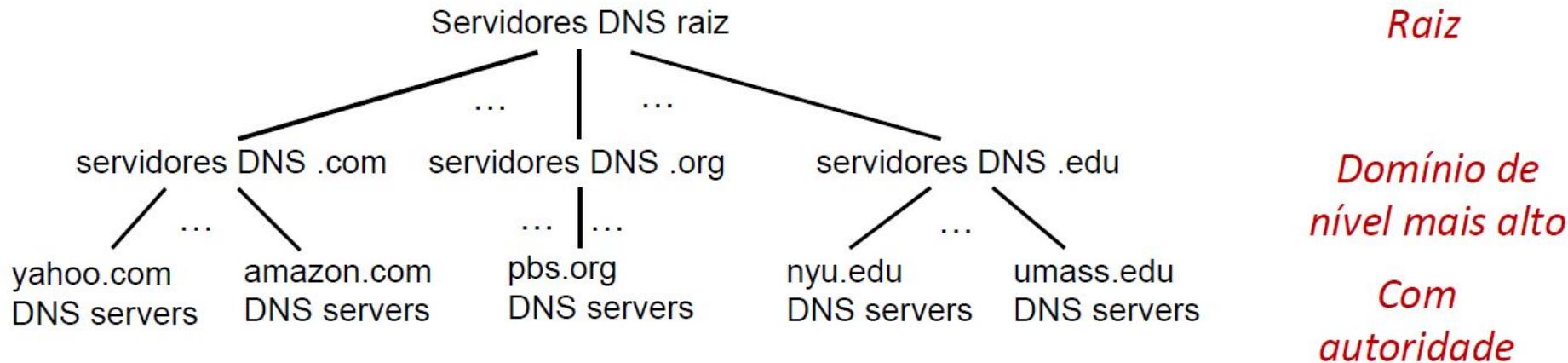
- muito mais leituras que escritas
- desempenho importa: quase todas as transações de Internet interagem com DNS - milissegundos contam!

**Organizacionalmente e fisicamente descentralizado:**

▪ milhões de organizações diferentes responsáveis por seus registros

**“a prova de balas”:** confiabilidade, segurança

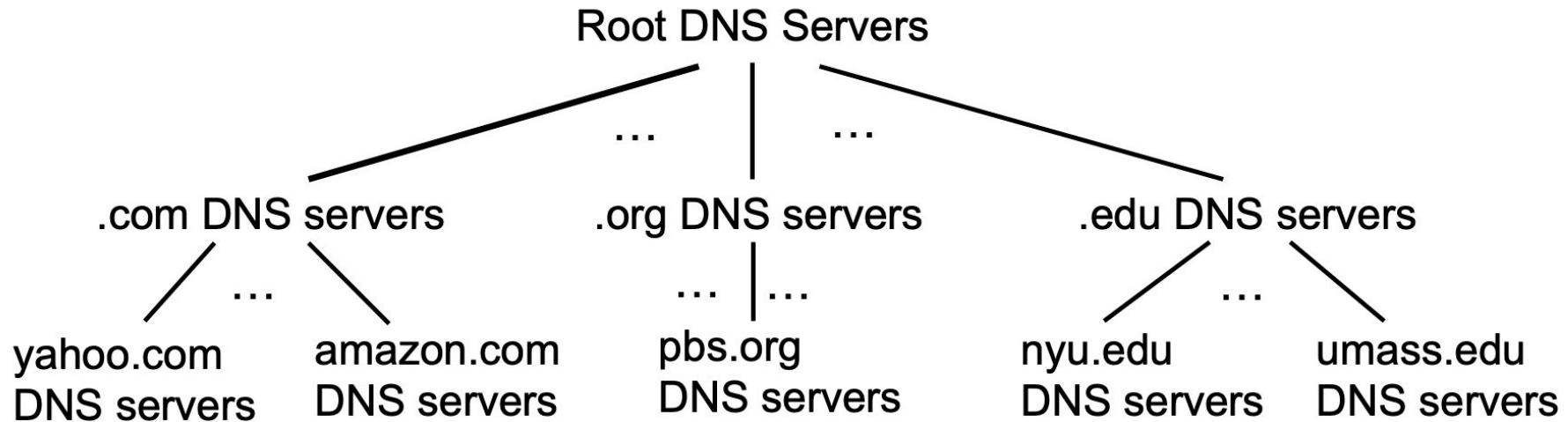
# DNS: um banco de dados distribuído e hierárquico



O cliente quer endereço IP para [www.amazon.com](http://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](http://www.amazon.com)

# DNS: servidores de nome raiz

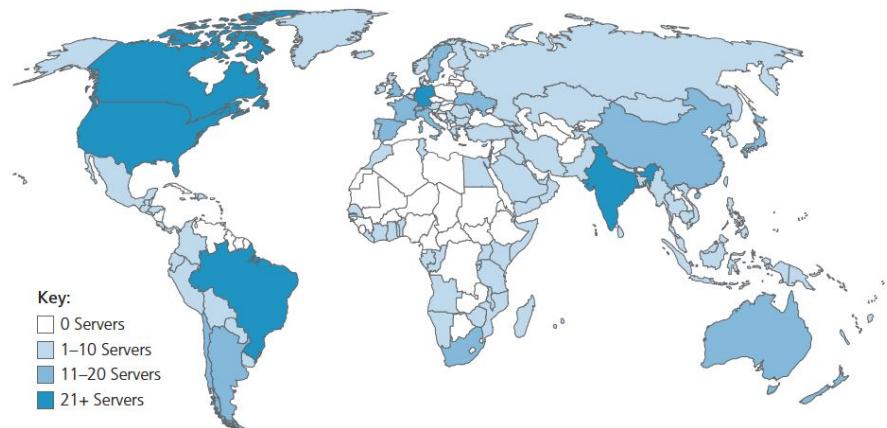


<https://root-servers.org/>

# 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) gerência domínio DNS raiz

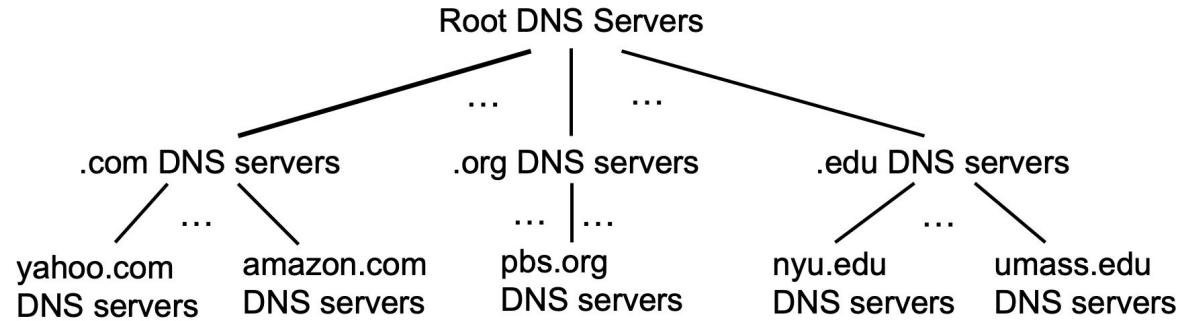
**13 “servidores” de nome raiz lógicos em todo o mundo, cada "servidor" replicado muitas vezes (~200 servidores nos EUA)**



# 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



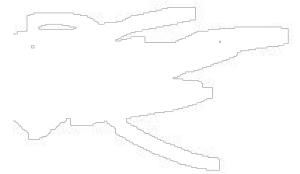
- servidores DNS com autoridade:
- servidores DNS da própria organização, fornecendo mapeamentos de nome de hospedeiro para IP com autoridade para os hospedeiros nomeados da organização
- pode ser mantido pela organização ou provedor de serviços  
<https://www.iana.org/domains/root/db>

# 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

# Resolução de nomes do DNS: consulta iterativa

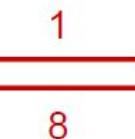
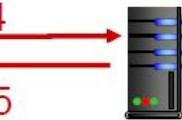
**Exemplo:** hospedeiro em engineering.nyu.edu quer o endereço IP para gaia.cs.umass.edu



servidor DNS raiz



servidor DNS TLD



servidor DNS local  
*dns.nyu.edu*

## Consulta iterativa:

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



hospedeiro  
requisitante em  
*engineering.nyu.edu*

servidor DNS com autoridade  
*dns.cs.umass.edu*

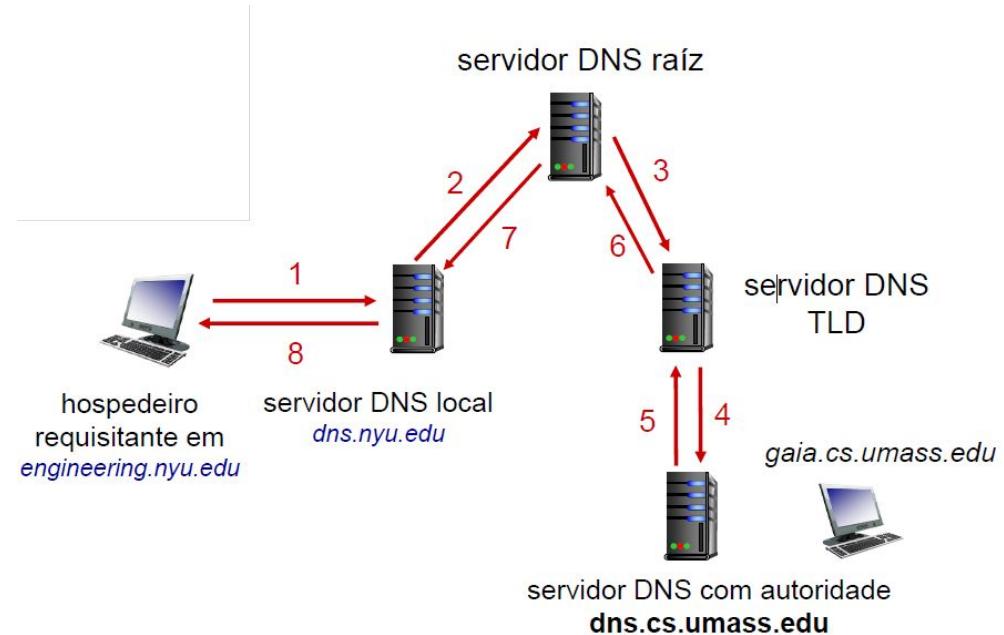


# Resolução de nomes do DNS: consulta recursiva

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?



# 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

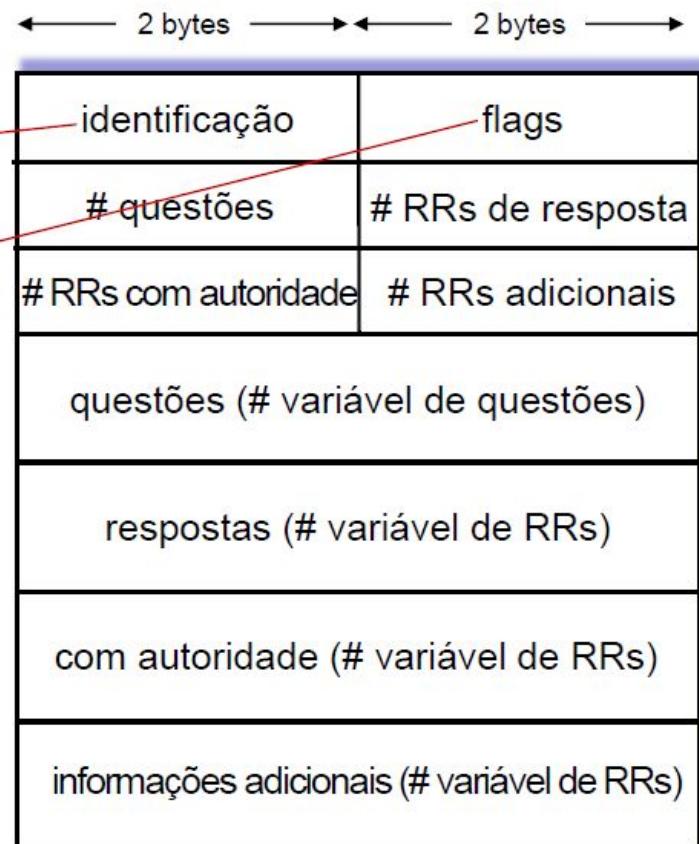
- valor é o nome de um servidor de SMTP (e-mail) associado com nome

# Mensagens do protocolo DNS

Mensagens de *requisição* e *resposta* do DNS tem o mesmo formato:

cabeçalho da mensagem:

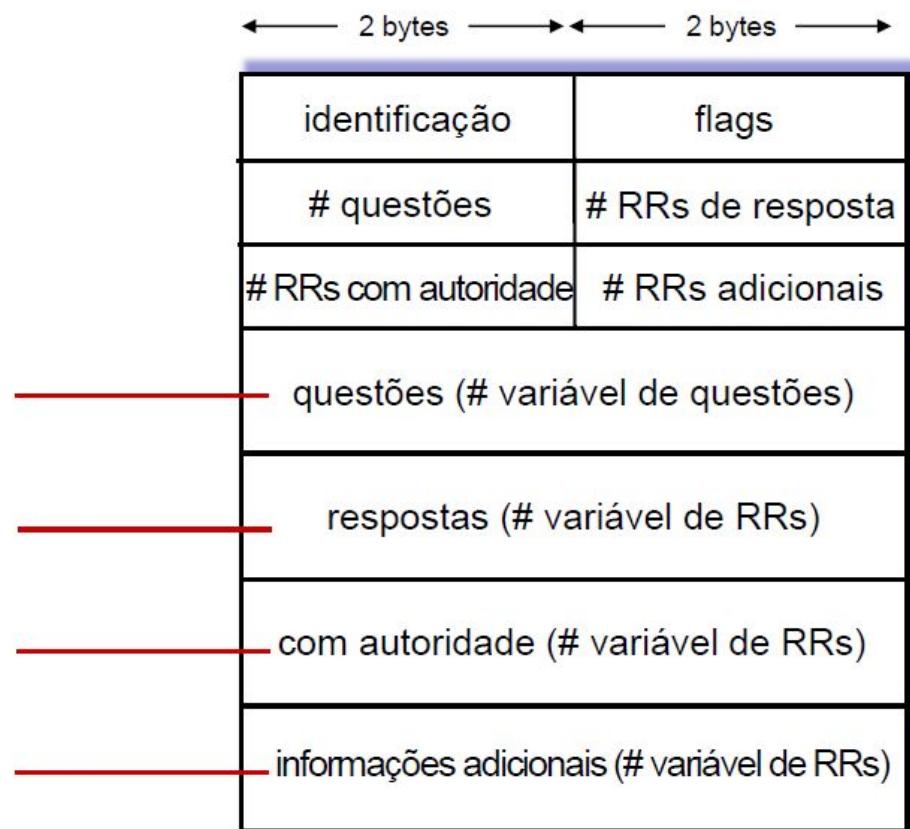
- identificação: número de 16 bit para requisição, resposta usa mesmo número
- flags:
  - requisição ou resposta
  - recursividade desejada
  - recursividade disponível
  - resposta é com autoridade



# Mensagens do protocolo DNS

Mensagens de *requisição* e *resposta* do DNS tem o mesmo formato:

- campos de nome e tipo para uma consulta
- RRs em resposta à requisição
- registros para servidores com autoridade
- informações adicionais “úteis” que podem ser usadas



# 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

# 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

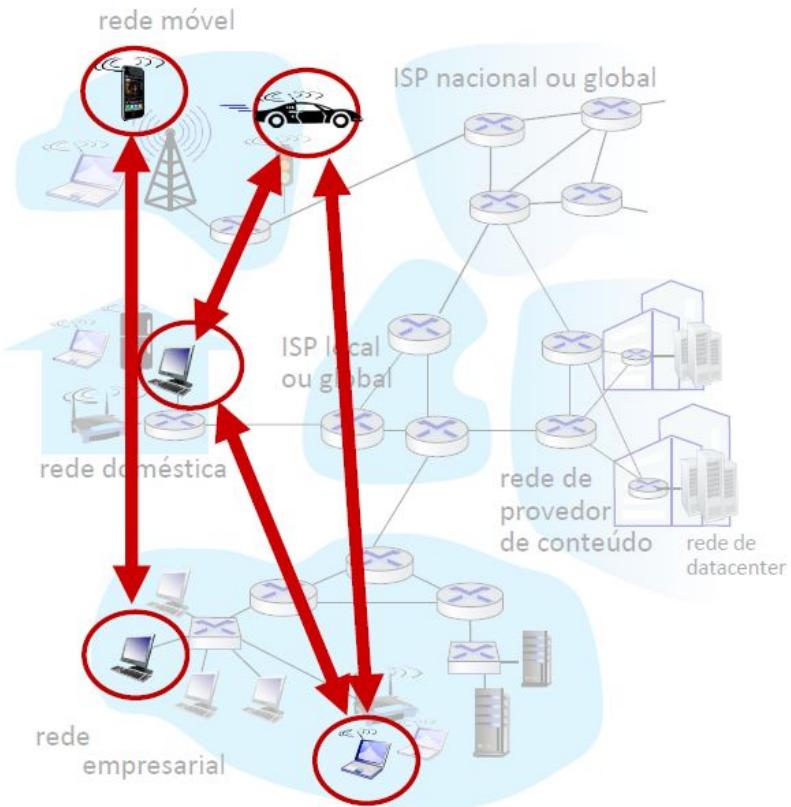
- interceptar consultas DNS, retornando respostas falsas
  - envenenamento de cache de DNS
  - RFC 4033: serviços de autenticação DNSSEC

# Camada de Aplicações

- Princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O Sistema de Nomes de Domínio (DNS)
- **Aplicações P2P**
- Transmissão de vídeo e redes de distribuição de conteúdo

# Arquitetura Peer-to-peer (P2P)

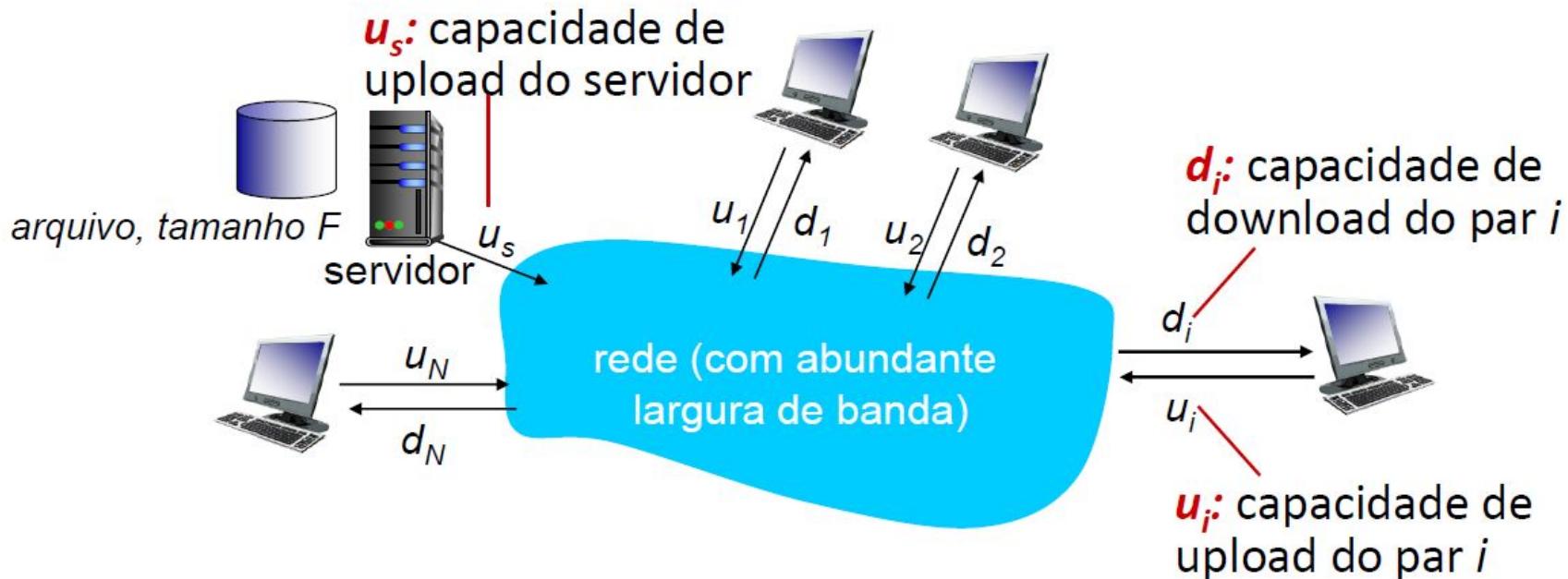
- sem servidor sempre ligado
- sistemas finais arbitrários se comunicam diretamente■ pares solicitam serviço de outros pares, e em troca oferecem serviços para outros pares
  - auto escalabilidade – novos pares trazem nova capacidade de atendimento e novas demandas de serviços
- os pares estão intermitentemente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



# Tempo de distribuição de arquivos: cliente-servidor

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

capacidade de upload/download por pares é recurso limitado



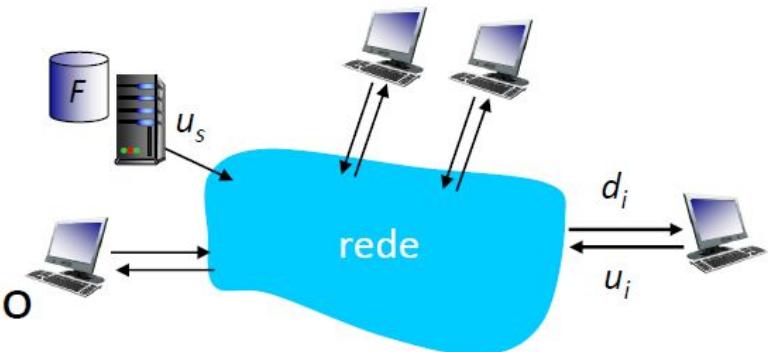
# Tempo de distribuição de arquivos: cliente-servidor

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

aumenta linearmente em  $N$

# Tempo de distribuição de arquivo: P2P

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

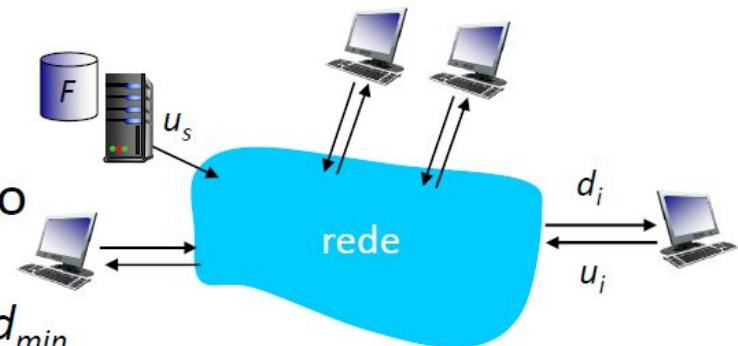
- tempo para enviar uma cópia:  $F/u_s$

- *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

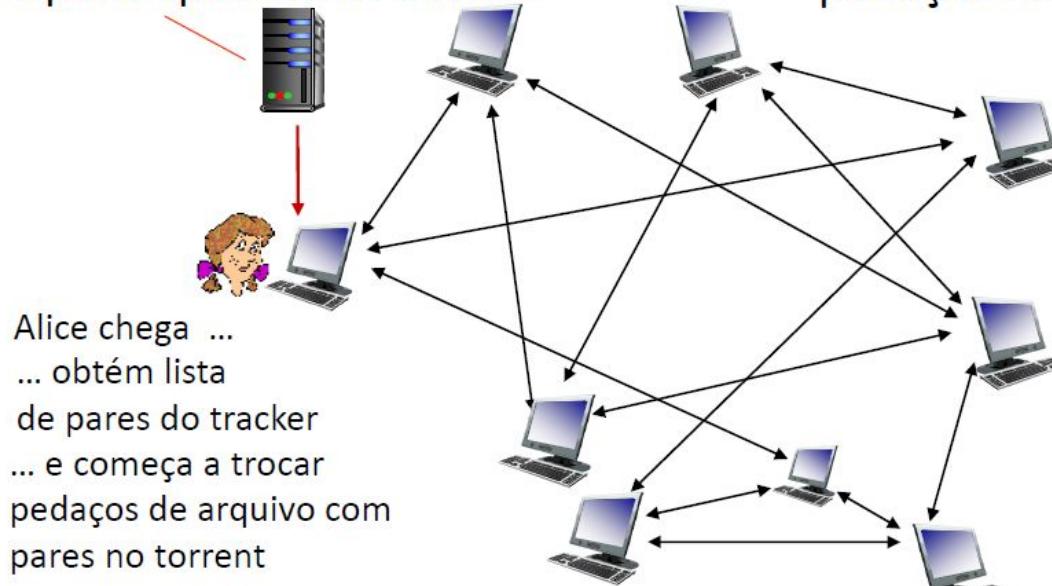
$$D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

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

# BitTorrent: solicitando, enviando pedaços de arquivo

## Solicitando pedaços:

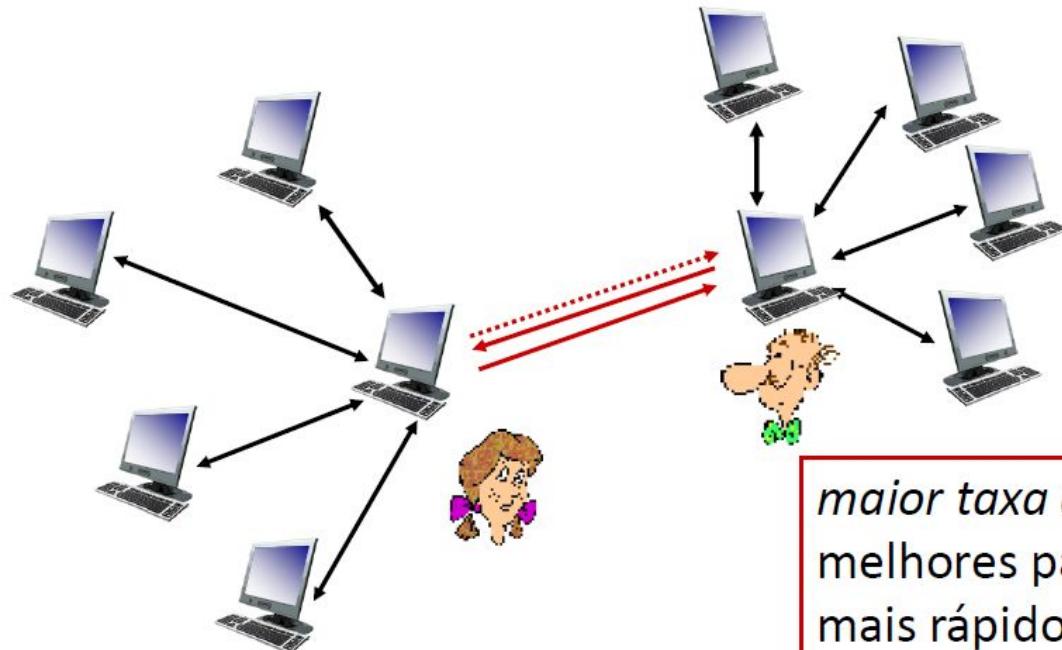
- a qualquer momento, diferentes pares têm subconjuntos diferentes de pedaços de arquivo
- periodicamente, Alice pede a cada par para que enviem uma lista dos pedaços que eles têm
- Alice pede pedaços faltantes para os pares, os mais raros primeiro

## 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 Aplicações

- Princípios de aplicações de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O Sistema de Nomes de Domínio (DNS)
- Aplicações P2P
- Transmissão de vídeo e redes de distribuição de conteúdo

# 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)

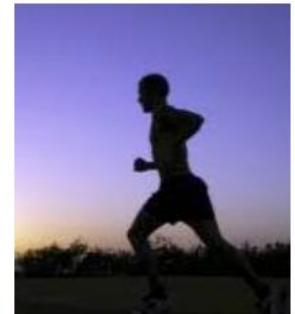
*exemplo de codificação espacial:*

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  $i$

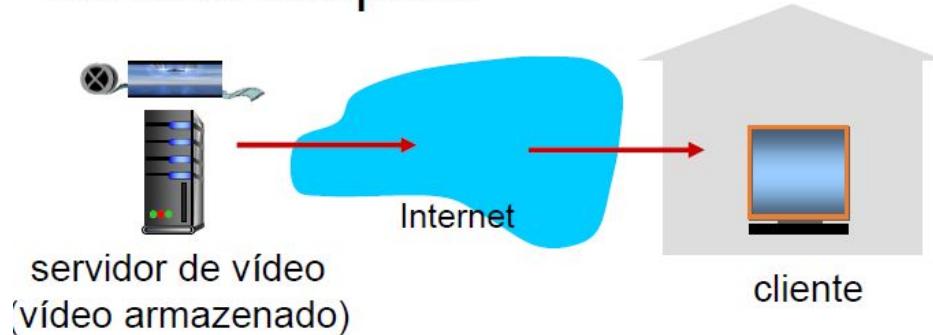
*exemplo de codificação temporal:* em vez de enviar quadro completo em  $i+1$ , enviar apenas diferenças dele para o quadro  $i$



quadro  $i+1$

# Transmissão de vídeo armazenado

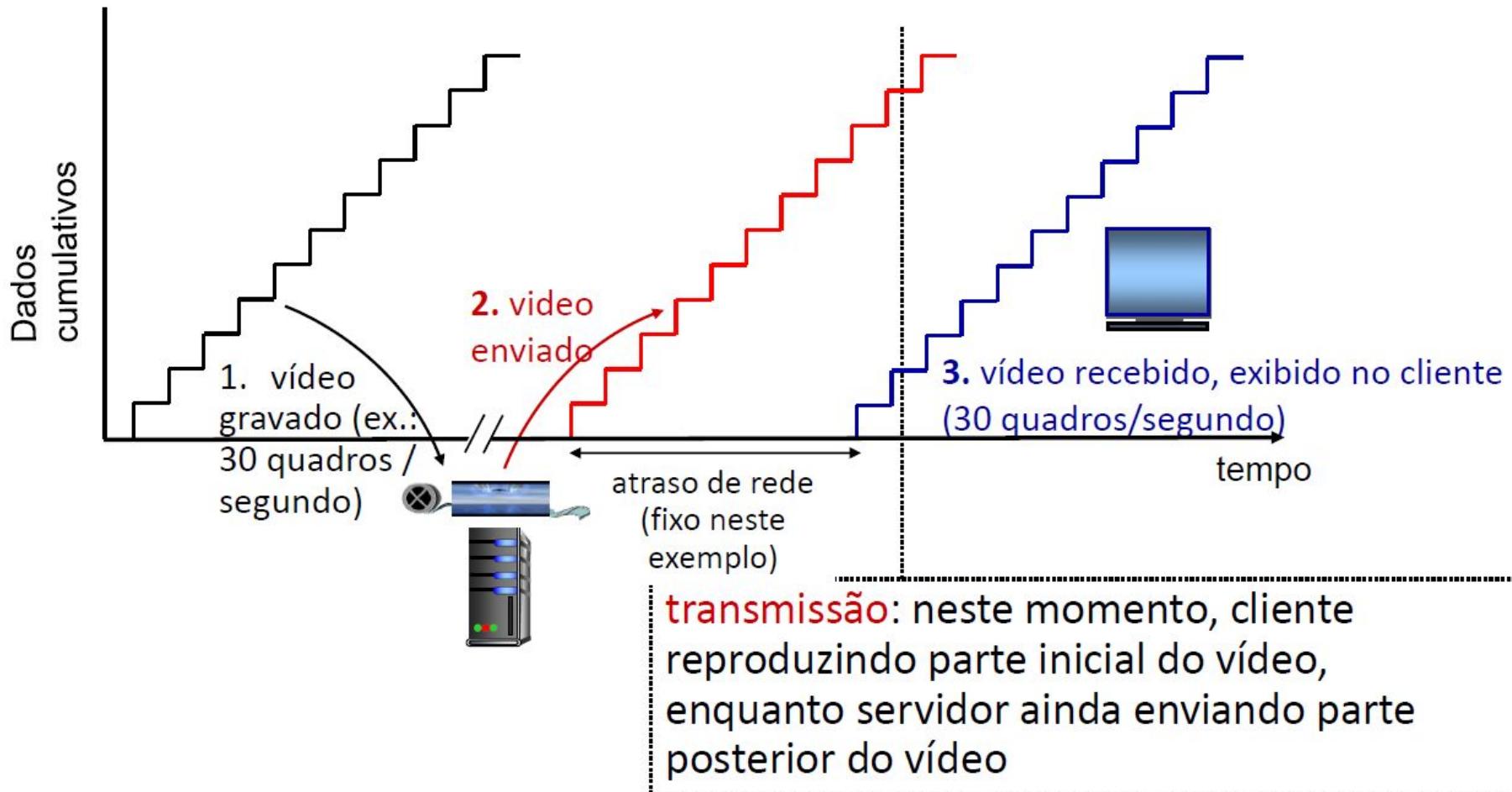
cenário simples:



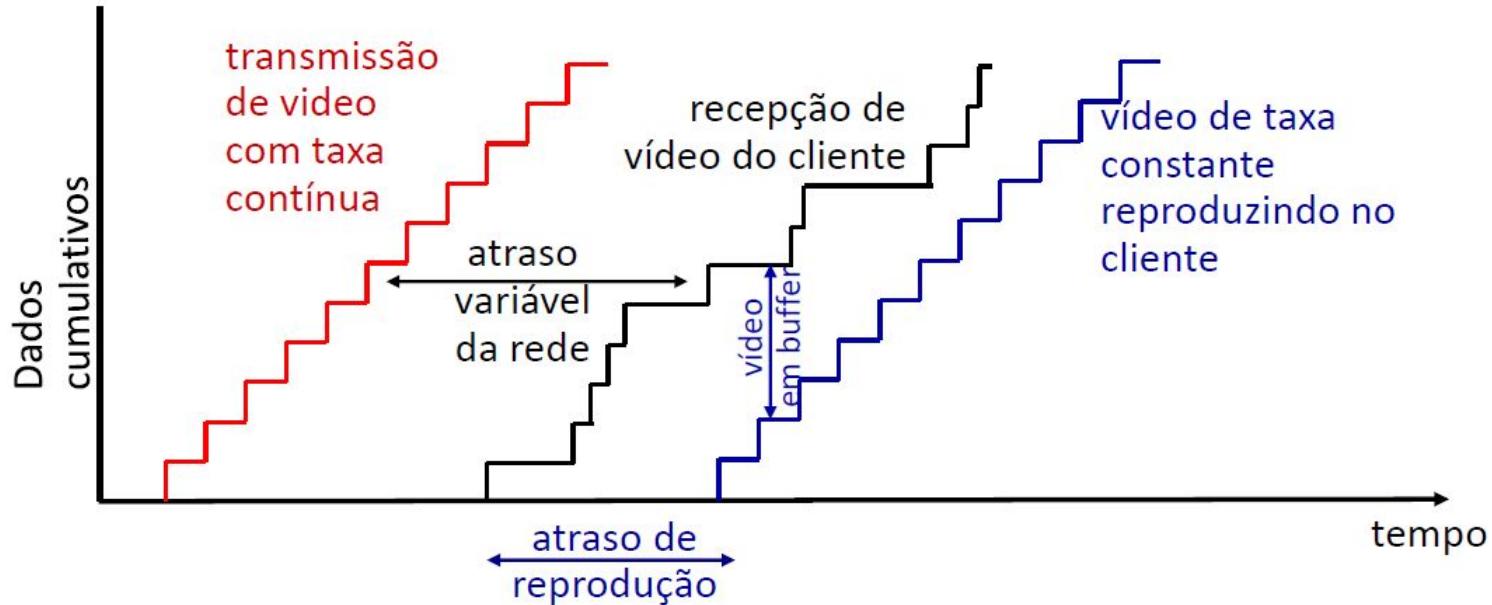
Principais desafios:

- 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: buffer de reprodução



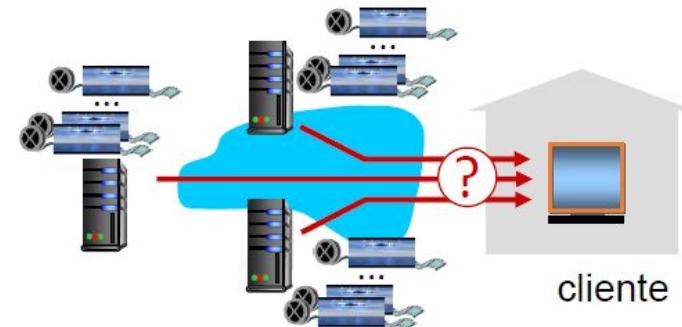
- **buffer do lado do cliente e atraso de reprodução:** compensa o atraso adicionado pela rede e a variação de atraso (*jitter*)

# Transmissão de vídeo armazenado: 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



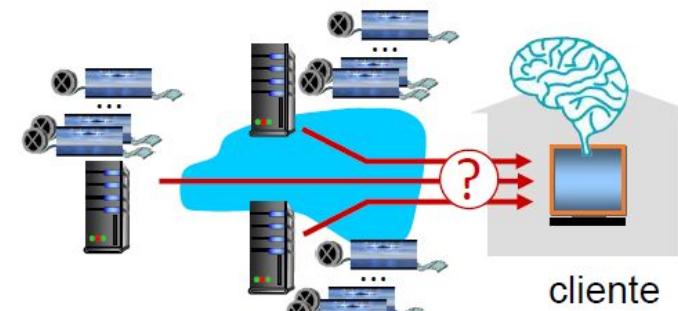
## cliente:

- 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 de vídeo armazenado: 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 “megaseridor”
  - 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

## Akamai Connected Cloud

A plataforma mais distribuída do mundo para serviços de computação em nuvem, segurança e entrega de conteúdo.



**4,100+**

PoPs de edge



**1,200+**

redes



**130+**

países



**1+ Pbps**

capacidade de edge



**24/7**

monitoramento

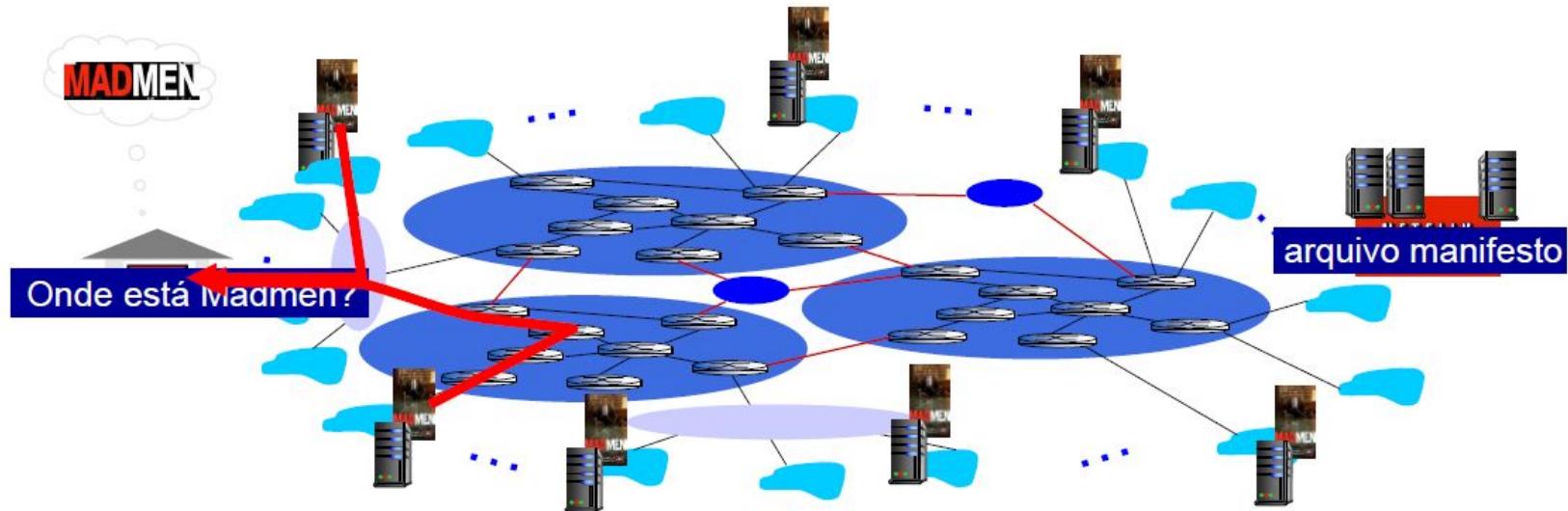


**2,000+**

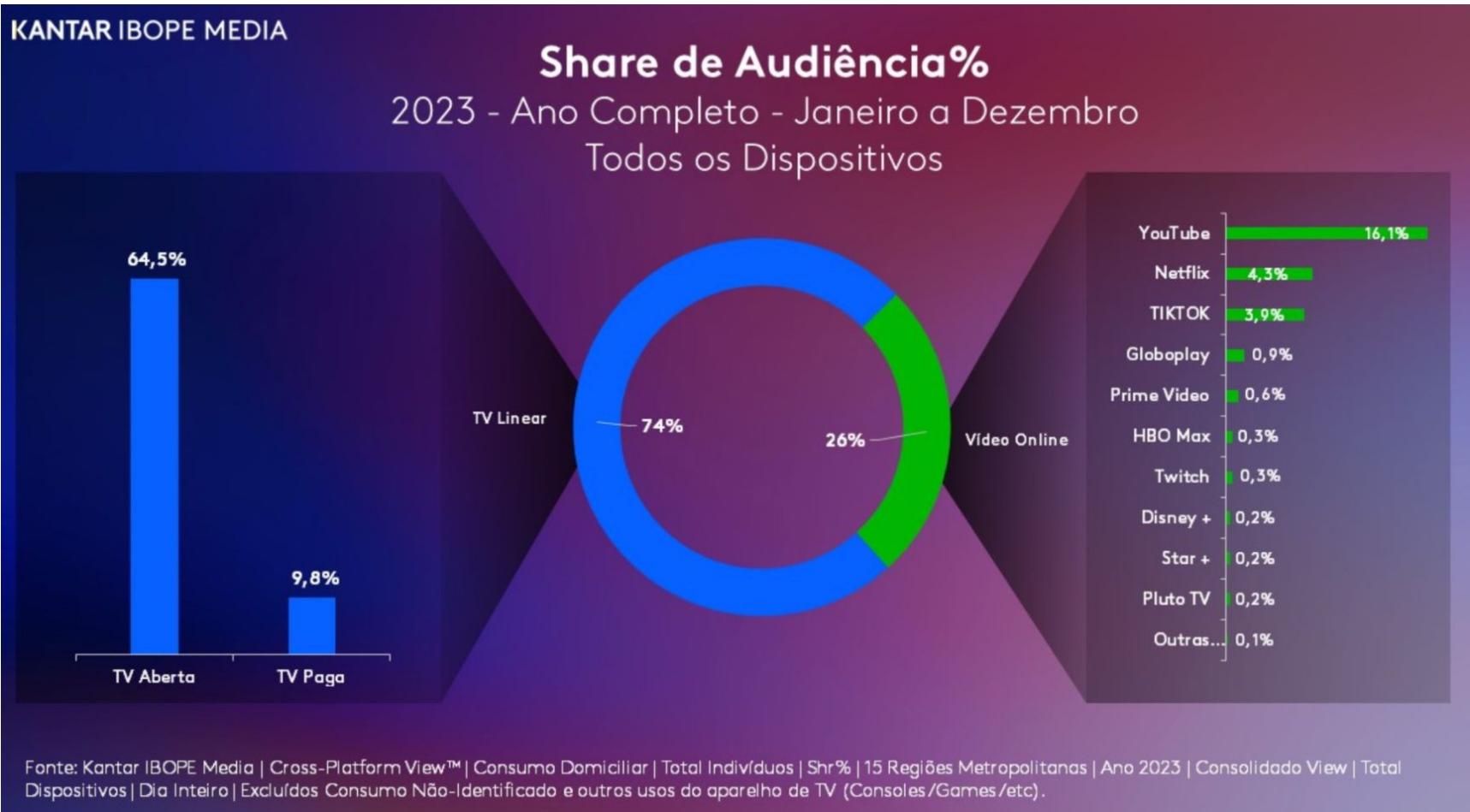
especialistas em serviços

# 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



# Audiência



<https://www.convergenciadigital.com.br/media/ilu/share-audiencia-streaming-tv-2023.jpg>