

Capítulo 2

Camada de aplicativos

Uma observação sobre o uso desses slides do PowerPoint:

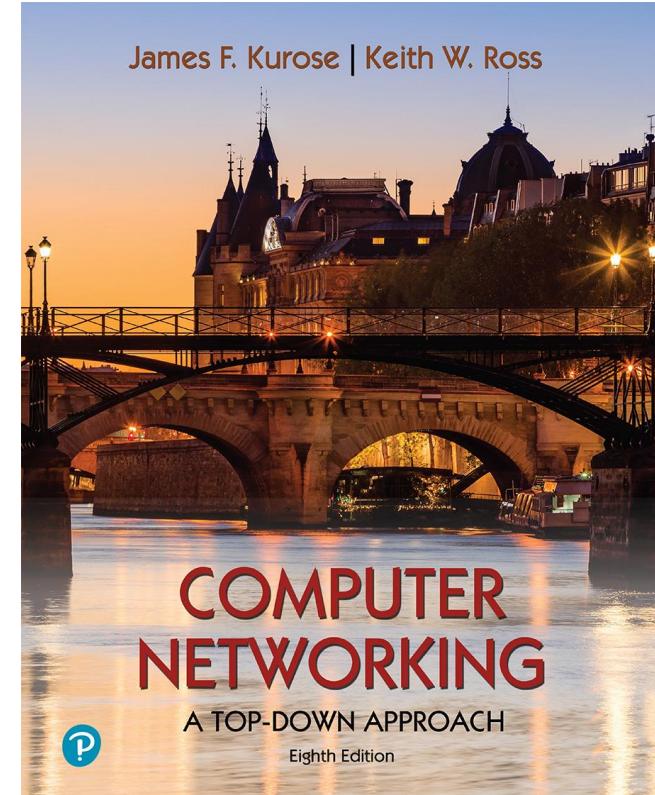
Estamos disponibilizando esses slides gratuitamente para todos (professores, alunos, leitores). Eles estão no formato PowerPoint para que você veja as animações e possa adicionar, modificar e excluir slides (inclusive este) e o conteúdo dos slides para atender às suas necessidades. Obviamente, eles representam *muito* trabalho de nossa parte. Em troca do uso, pedimos apenas o seguinte:

- Se você usar esses slides (por exemplo, em uma aula), mencione a fonte (afinal, gostaríamos que as pessoas usassem nosso livro!)
- Se você publicar algum slide em um site www, informe que ele foi adaptado de nossos slides (ou talvez idêntico a eles) e informe nossos direitos autorais sobre esse material.

Para obter um histórico de revisões, consulte a nota do slide desta página.

Obrigado e divirta-se! JFK/KWR

Todos os materiais têm direitos autorais de 1996 a 2023
J.F. Kurose e K.W. Ross, Todos os direitos reservados



Redes de computadores: A Top-Down Approach (Uma abordagem de cima para baixo)

8th edição n
Jim Kurose, Keith Ross
Pearson, 2020

Camada de aplicativos: visão geral

- Princípios de aplicativos de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O sistema de nomes de domínio DNS

- Aplicativos P2P
- redes de distribuição de conteúdo e streaming de vídeo
- programação de soquetes com UDP e TCP



Camada de aplicativos: visão geral

Nossos objetivos:

- aspectos conceituais e de implementação de protocolos de camada de aplicativos
 - modelos de serviço da camada de transporte
 - paradigma cliente-servidor
 - paradigma ponto a ponto
- aprender sobre protocolos examinando protocolos populares de camada de aplicativos e infraestrutura
 - HTTP
 - SMTP, IMAP
 - DNS
 - sistemas de streaming de vídeo, CDNs
- programação de aplicativos de rede
 - API de soquete

Alguns aplicativos 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 (por exemplo, Skype)
 - videoconferência em tempo real (por exemplo, Zoom)
 - Pesquisa na Internet
 - login remoto
 - ...
- P: Seus favoritos?*

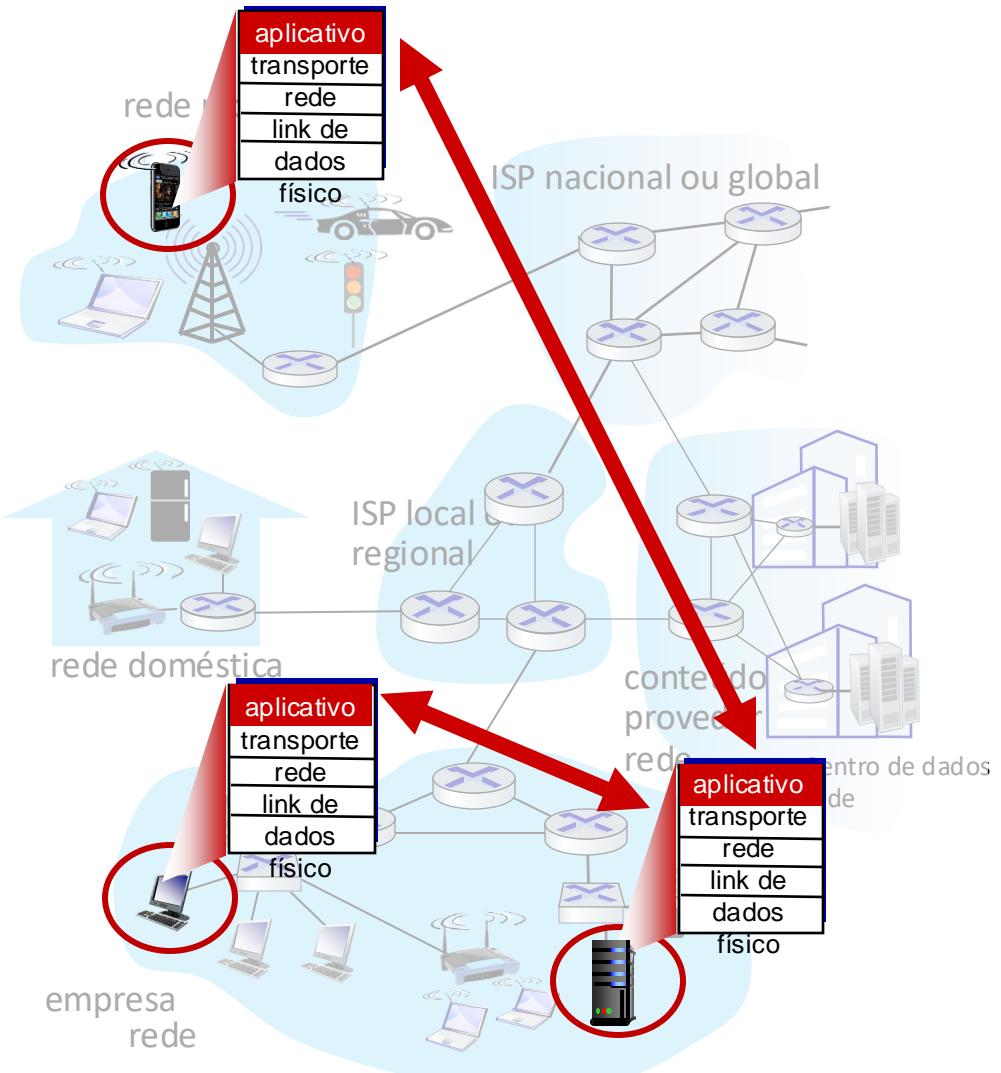
Criando um aplicativo de rede

escrever programas que:

- executados em sistemas finais (diferentes)
- comunicar-se pela rede
- Por exemplo, o software do servidor da Web se comunica com o software do navegador

não há necessidade de escrever software para dispositivos com núcleo de rede

- os dispositivos com núcleo de rede não executam aplicativos de usuário
- aplicativos em sistemas finais permite o desenvolvimento rápido de



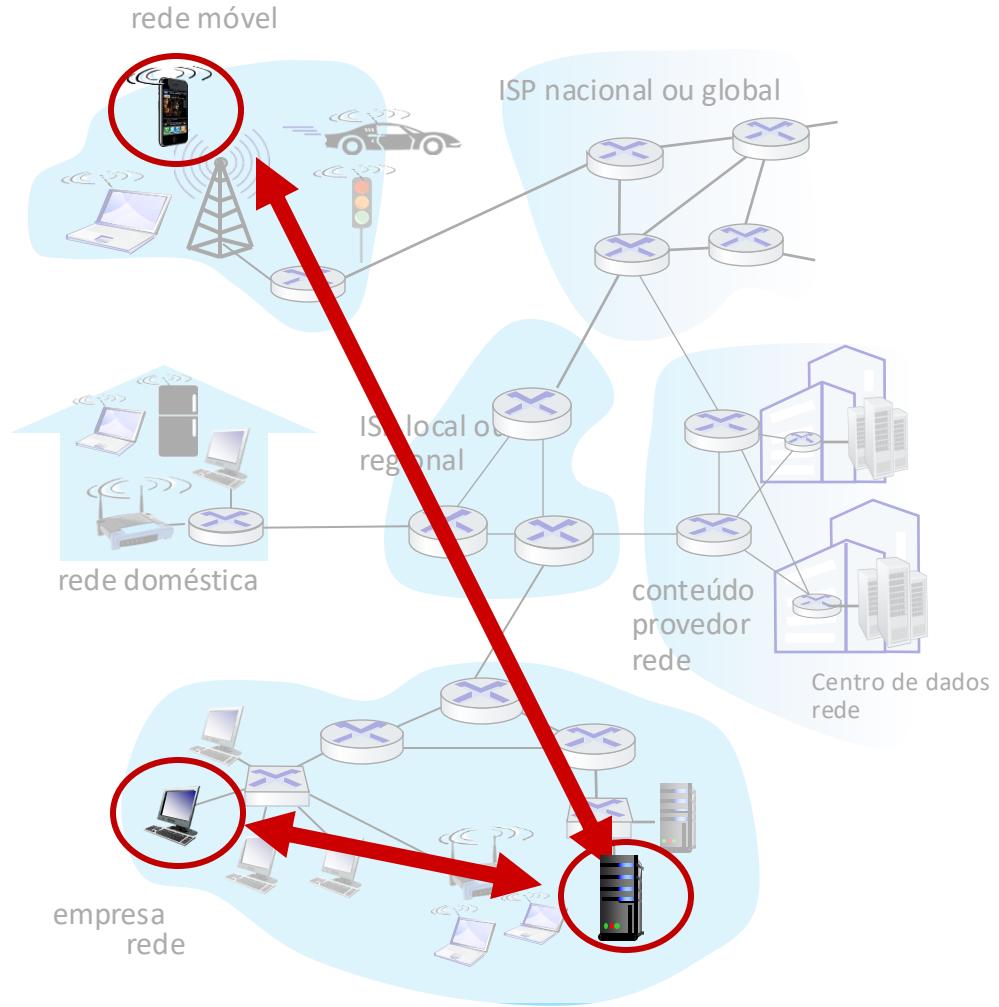
Paradigma cliente-servidor

servidor:

- host sempre ativo
- endereço IP permanente
- frequentemente em data centers, para dimensionamento

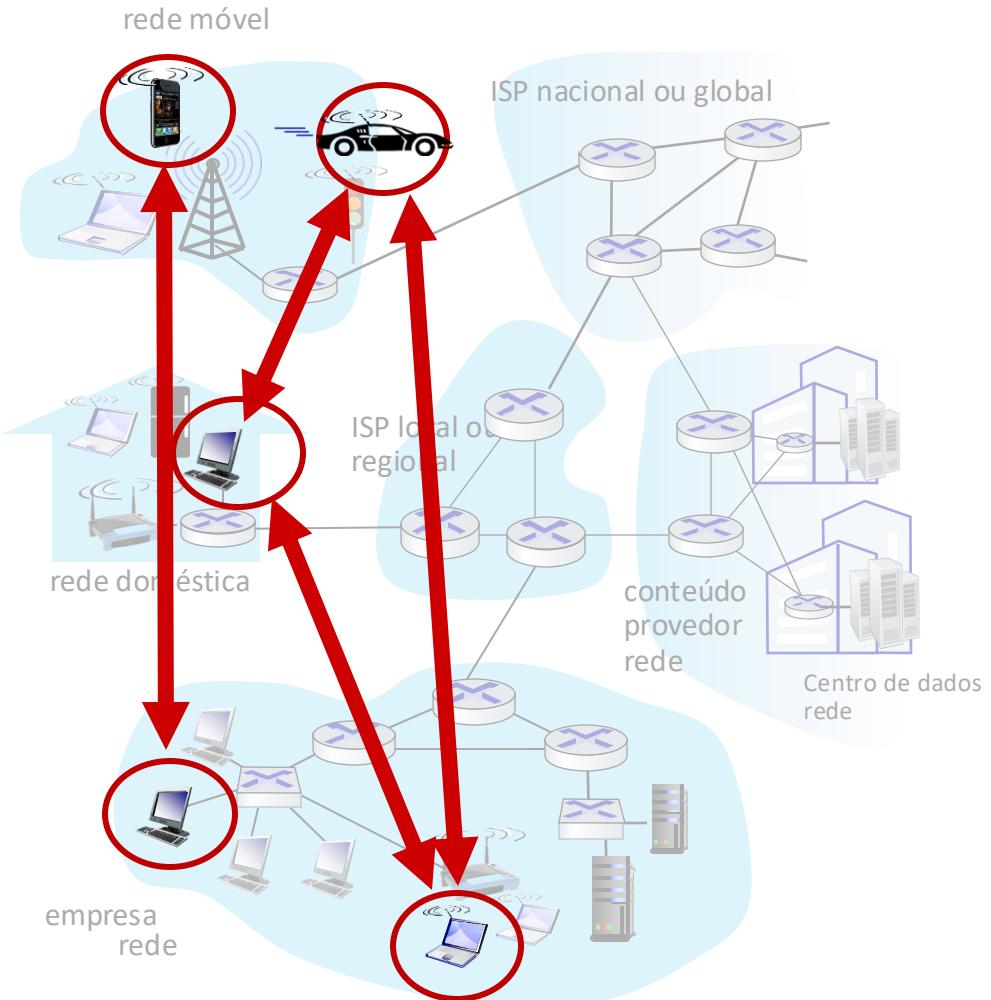
clientes:

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



Arquitetura ponto a ponto

- *nenhum* servidor sempre ativo
- sistemas finais arbitrários se comunicam diretamente
- os pares solicitam serviços de outros pares e fornecem serviços em troca a outros pares
 - *autoescalabilidade* - novos pares trazem nova capacidade de serviço, bem como novas demandas de serviço
- os pares estão conectados de forma intermitente e mudam os endereços IP
 - gerenciamento complexo
- exemplo: Compartilhamento de arquivos P2P [BitTorrent]



Processos de comunicação

processo: programa executado em um host

- No mesmo host, dois processos se comunicam usando a **comunicação entre processos** (definida pelo sistema operacional)
- Os processos em diferentes hosts se comunicam por meio da troca de **mensagens**

clientes, servidores

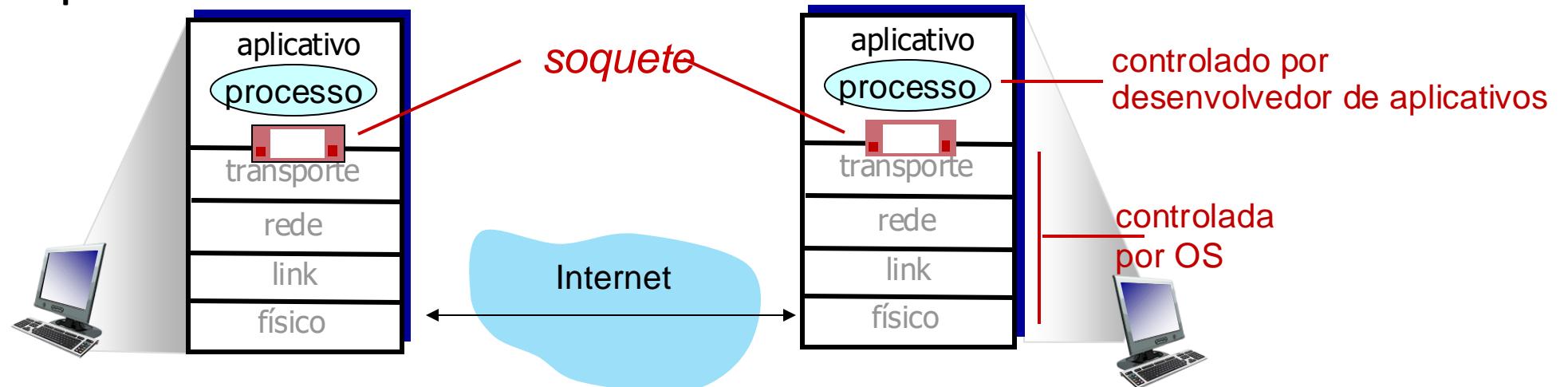
processo do cliente: processo que inicia a comunicação

processo servidor: processo que aguarda ser contatado

- Observação: os aplicativos com arquiteturas P2P têm processos de cliente e processos de servidor

Soquetes

- o processo envia/recebe mensagens de/para seu **soquete**
- soquete análogo à porta
 - o processo de envio empurra a mensagem para fora da porta
 - O processo de envio depende da infraestrutura de transporte do outro lado da porta para entregar a mensagem ao soquete no processo de recebimento
 - dois soquetes envolvidos: um em cada lado



Processos de endereçamento

- para receber mensagens, o processo deve ter *o identificador*
- o dispositivo host tem um endereço IP exclusivo de 32 bits
- P: O endereço IP do host no qual o processo é executado é suficiente para identificar o processo?
- R: não, muitos processos podem estar em execução no mesmo host
- *O identificador* inclui o endereço IP e os números de porta associados ao processo no host.
- números de porta de exemplo:
 - Servidor HTTP: 80
 - servidor de correio eletrônico: 25
- para enviar mensagem HTTP ao servidor da Web gaia.cs.umass.edu:
 - Endereço IP: 128.119.245.12
 - Número da porta: 80
- mais em breve...

Um protocolo de camada de aplicativo 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 os processos enviam e respondem a mensagens

protocolos abertos:

- definidos em RFCs, todos têm acesso à definição do protocolo
- permite a interoperabilidade
- Por exemplo, HTTP, SMTP

protocolos proprietários:

- Por exemplo, Skype, Zoom

De que serviço de transporte um aplicativo precisa?

integridade dos dados

- alguns aplicativos (por exemplo, transferência de arquivos, transações na Web) exigem uma transferência de dados 100% confiável
- outros aplicativos (por exemplo, áudio) podem tolerar alguma perda

tempo

- alguns aplicativos (por exemplo, telefonia pela Internet, jogos interativos) exigem baixo atraso para serem "eficazes"

rendimento

- alguns aplicativos (por exemplo, multimídia) exigem uma quantidade mínima de throughput para serem "eficazes"
- outros aplicativos ("aplicativos elásticos") fazem uso de qualquer taxa de transferência que obtêm

segurança

- criptografia, integridade de dados, ...

Requisitos de serviço de transporte: aplicativos comuns

aplicativo	perda de dados?	rendimento	sensível ao tempo?
transferência de arquivos	sem perda	elástico	não
e-mail	sem perda	elástico	não
Documentos da Web	sem perda	elástico	não
áudio/vídeo em tempo real	tol. a perdas	áudio: 5Kbps-1Mbps vídeo:10Kbps-5Mbps	Sim, 10 msec
streaming de áudio/vídeo	tol. a perdas	igual ao anterior	sim, alguns segundos
jogos interativos	tol. a perdas	Kbps+	Sim, 10 msec
mensagens de texto	sem perda	elástico	sim e não

Serviços de protocolos de transporte da Internet

Serviço TCP:

- *transporte confiável* entre os processos de envio e recebimento
- *controle de fluxo*: o remetente não sobrecarrega o receptor
- *controle de congestionamento*: controle do remetente quando a rede está sobrecarregada
- *Orientado à conexão*: configuração necessária entre os processos do cliente e do servidor
- *não fornece*: tempo, garantia de taxa de transferência mínima, segurança

Serviço UDP:

- *transferência de dados não confiável* entre os processos de envio e recebimento
- *não oferece*: confiabilidade, controle de fluxo, controle de congestionamento, tempo, garantia de throughput, segurança ou configuração de conexão.

P: por que se preocupar?
Por que existe um UDP?

Aplicativos da Internet e protocolos de transporte

aplicativo	aplicativo	protocolo de camada	protocolo de transporte
transferência de arquivos	FTP [RFC 959]		TCP
e-mail	SMTP [RFC 5321]		TCP
Documentos da Web	HTTP [RFC 7230, 9110]		TCP
Telefonia via Internet	SIP [RFC 3261], RTP [RFC 3550] ou HTTP proprietário		TCP ou UDP
streaming de áudio/vídeo	[RFC 7230], DASH		TCP
jogos interativos	WOW, FPS (proprietário)		UDP ou TCP

Protegendo o TCP

Sockets Vanilla TCP e UDP:

- sem criptografia
- senhas em texto claro enviadas para o soquete atravessam a Internet em texto claro (!)

Segurança da camada de transporte (TLS)

- fornece conexões TCP criptografadas
- integridade dos dados
- autenticação de ponto final

TLS implementado na camada de aplicativos

- Os aplicativos usam bibliotecas TLS, que, por sua vez, usam TCP
- texto claro enviado para o "soquete" atravessa a Internet criptografado
- mais: Capítulo 8

Camada de aplicativos: visão geral

- Princípios de aplicativos de rede
- **Web e HTTP**
- E-mail, SMTP, IMAP
- O sistema de nomes de domínio DNS
- Aplicativos P2P
- redes de distribuição de conteúdo e streaming de vídeo
- programação de soquetes com UDP e TCP



Web e HTTP

Primeiro, uma rápida revisão...

- A página da Web consiste em *objetos*, cada um dos quais pode ser armazenado em diferentes servidores da Web
- O objeto pode ser um arquivo HTML, uma imagem JPEG, um applet Java, um arquivo de áudio,...
- A página da Web consiste em um *arquivo HTML básico* que inclui *vários objetos referenciados, cada um* endereçado por um *URL, por exemplo,*

www . someschool . edu / someDept / pic . gif

nome do host

nome do caminho

Visão geral do HTTP

HTTP: protocolo de transferência de hipertexto

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



Visão geral do HTTP (continuação)

O HTTP usa TCP:

- o cliente inicia a conexão TCP (cria um soquete) com o servidor, porta 80
- o servidor aceita a conexão TCP do cliente
- Mensagens HTTP (mensagens de protocolo de camada de aplicativo) trocadas entre o navegador (cliente HTTP) e o servidor da Web (servidor HTTP)
- Conexão TCP fechada

O HTTP é "sem estado"

- o servidor *não* mantém informações sobre solicitações anteriores do cliente

à parte

Os protocolos que mantêm o "estado" são complexos!

- o histórico anterior (estado) deve ser mantido
- se o servidor/cliente falhar, suas visões de "estado" podem ser inconsistentes e 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

O download de vários objetos exigia várias conexões

HTTP persistente

- Conexão TCP aberta com um servidor
- vários objetos podem ser enviados em *uma única* conexão TCP entre o cliente e o servidor
- Conexão TCP fechada

HTTP não persistente: exemplo

O usuário insere o URL: `www.someSchool.edu/someDepartment/home.index`

(contendo texto, referências a 10 imagens jpeg)



1a. O cliente HTTP inicia a conexão TCP com o servidor HTTP (processo) em `www.someSchool.edu` na porta 80



1b. Servidor HTTP no host `www.someSchool.edu` aguardando conexão TCP na porta 80 "aceita" a conexão, notificando o cliente

tempo
↓

2. O cliente HTTP envia *uma mensagem de solicitação* HTTP (contendo URL) para o soquete de conexão TCP. A mensagem indica que o cliente deseja o objeto `someDepartment/home.index`

3. O servidor HTTP recebe a mensagem de solicitação, forma *uma mensagem de resposta* contendo o objeto solicitado e envia a mensagem para o seu soquete

HTTP não persistente: exemplo (cont.)

O usuário insere o URL: `www.someSchool.edu/someDepartment/home.index`
(contendo texto, referências a 10 imagens jpeg)



- tempo ↓
1. O usuário insere o URL: `www.someSchool.edu/someDepartment/home.index`
 2. O cliente HTTP envia uma mensagem de solicitação para o servidor HTTP.
 3. O servidor HTTP processa a solicitação e gera uma resposta.
 4. O servidor HTTP envia a resposta para o cliente HTTP.
 5. O cliente HTTP recebe uma mensagem de resposta contendo um arquivo html e exibe o html. Analisando o arquivo html, encontra 10 objetos jpeg referenciados
 6. As etapas de 1 a 5 são repetidas para cada um dos 10 objetos jpeg

4. O servidor HTTP fecha a conexão TCP.



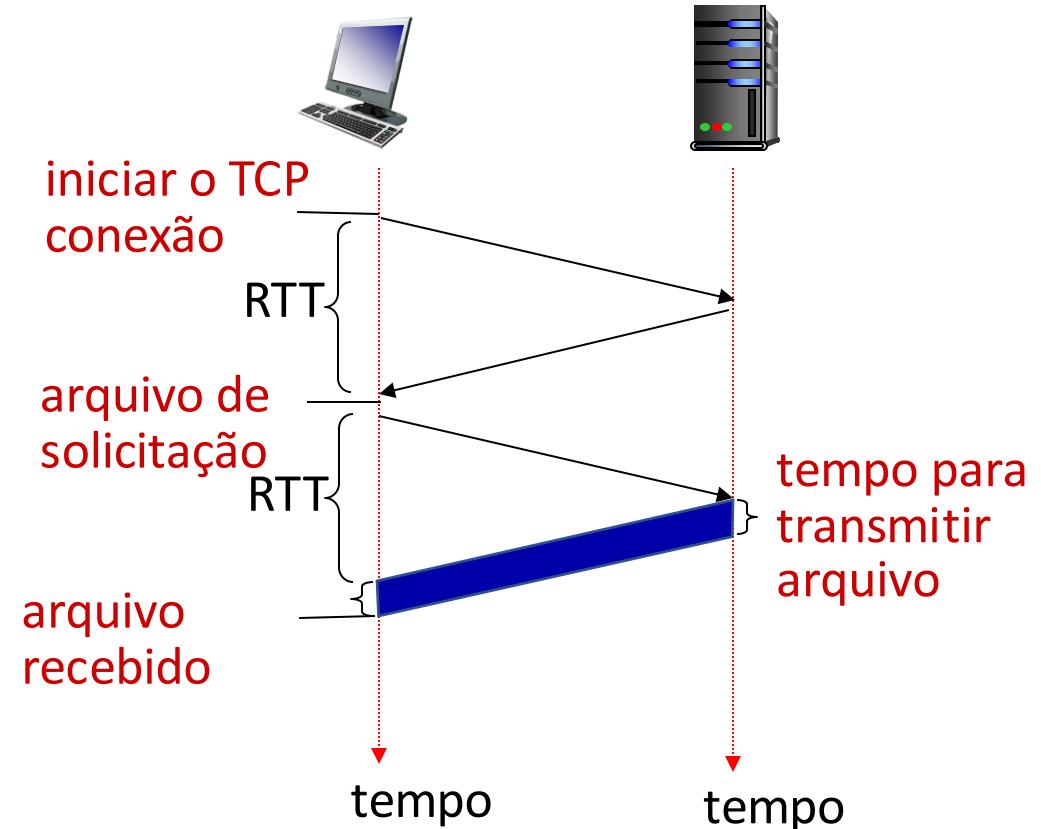
HTTP não persistente: tempo de resposta

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

Tempo de resposta HTTP (por objeto):

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

Tempo de resposta HTTP não persistente = 2RTT + tempo de transmissão do arquivo



HTTP persistente (HTTP 1.1)

Problemas de HTTP não persistentes:

- requer 2 RTTs por objeto
- Sobre carga do sistema operacional para *cada* conexão TCP
- Os navegadores geralmente abrem várias conexões TCP paralelas para buscar objetos referenciados em paralelo

HTTP persistente (HTTP1.1):

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

Mensagem de solicitação HTTP

- dois tipos de mensagens HTTP: *solicitação, resposta*
- Mensagem de solicitação HTTP:
 - ASCII (formato legível por humanos)

linha de solicitação

(GET, POST,

Comandos HEAD)

Cá

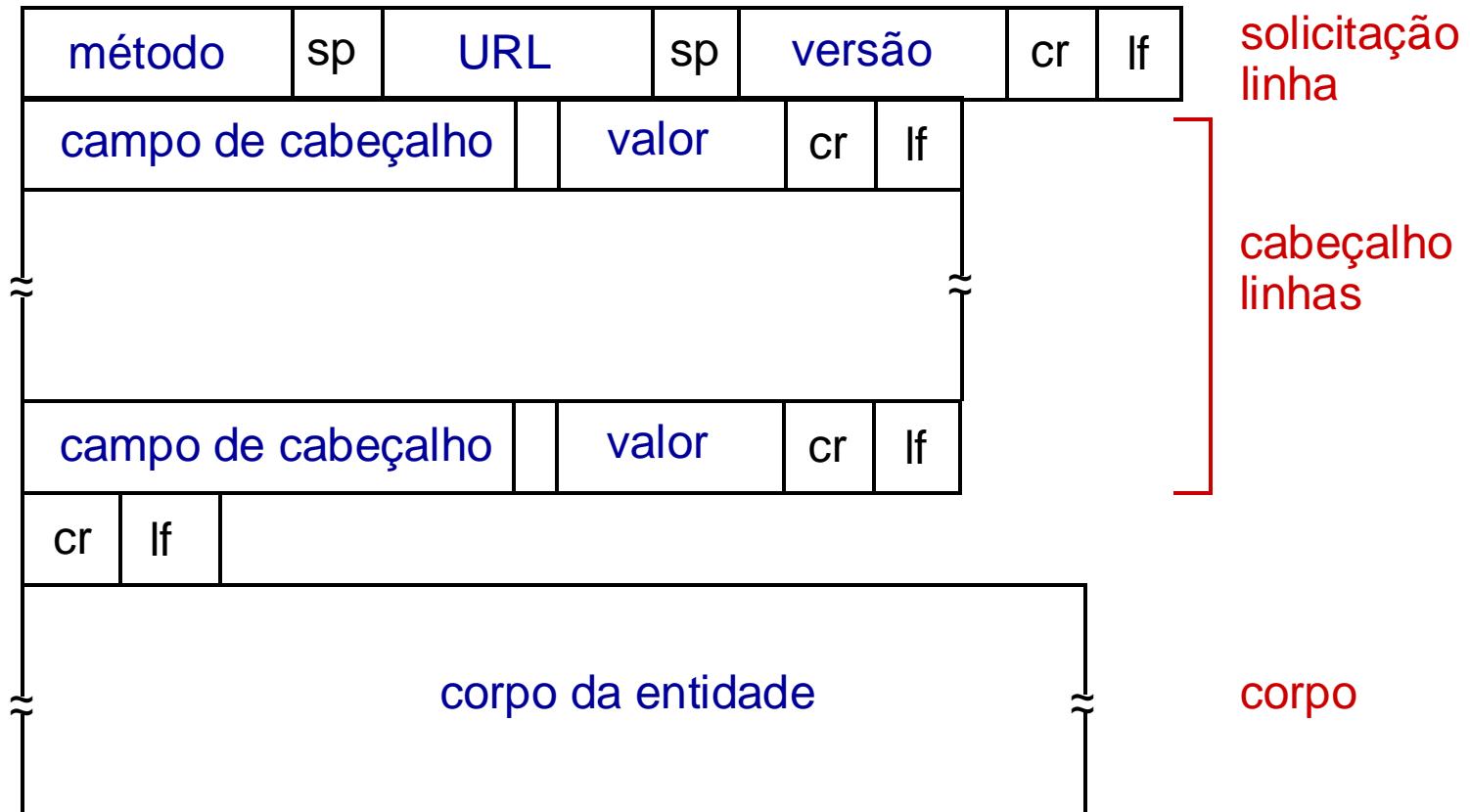
caractere de retorno de carro
caractere de alimentação de linha

retorno de carro, avanço →

de linha no início da
linha indica o fim das
linhas de cabeçalho

* Confira os exercícios interativos on-line para obter
mais exemplos:
http://gaia.cs.umass.edu/kurose_ross/interactive/

Mensagem de solicitação HTTP: formato geral



Outras mensagens de solicitação HTTP

Método POST:

- A página da Web geralmente inclui entradas de formulário
- entrada do usuário enviada do cliente para o servidor no corpo da entidade da mensagem de solicitação HTTP POST

Método GET (para enviar dados ao servidor):

- incluir dados do usuário no campo URL da mensagem de solicitação HTTP GET (após um '?'):

`www.somesite.com/animalsearch?monkeys&banana`

Método HEAD:

- solicita cabeçalhos (somente) que seriam retornados se o URL especificado fosse solicitado com um método HTTP GET.

Método PUT:

- faz o upload de um novo arquivo (objeto) para o servidor
- substitui completamente o arquivo existente no URL especificado pelo conteúdo no corpo da entidade da mensagem de solicitação HTTP POST

Mensagem de resposta HTTP

linha de status (protocolo → **HTTP/1.1 200 OK**
código de status frase de
status)

C.

Códigos de status de resposta HTTP

- O código de status aparece na primeira linha da mensagem de resposta de servidor para cliente.
- alguns códigos de amostra:

200 OK

- solicitação bem-sucedida, objeto solicitado mais adiante nesta mensagem

301 Movido permanentemente

- objeto solicitado movido, novo local especificado posteriormente nesta mensagem (no campo Location:)

400 Solicitação incorreta

- mensagem de solicitação não compreendida pelo servidor

404 Não encontrado

- O documento solicitado não foi encontrado neste servidor

505 Versão HTTP não suportada

Experimentando o HTTP (lado do cliente) por conta própria

1. netcat para seu servidor da Web favorito:

```
% nc -c -v gaia.cs.umass.edu 80 (para Mac)
```

```
>ncat -C gaia.cs.umass.edu 80 (para Windows)
```

- abre conexão TCP com a porta 80 (porta padrão do servidor HTTP) em gaia.cs.umass.edu.
- Tudo o que for digitado será enviado para a porta 80 em gaia.cs.umass.edu

2. digite uma solicitação HTTP GET:

GET /kurose_ross/interactive/index.php HTTP/1.1

Host: gaia.cs.umass.edu

- Ao digitar isso (pressione duas vezes o retorno do carro), você envia essa solicitação GET mínima (mas completa) ao servidor HTTP

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

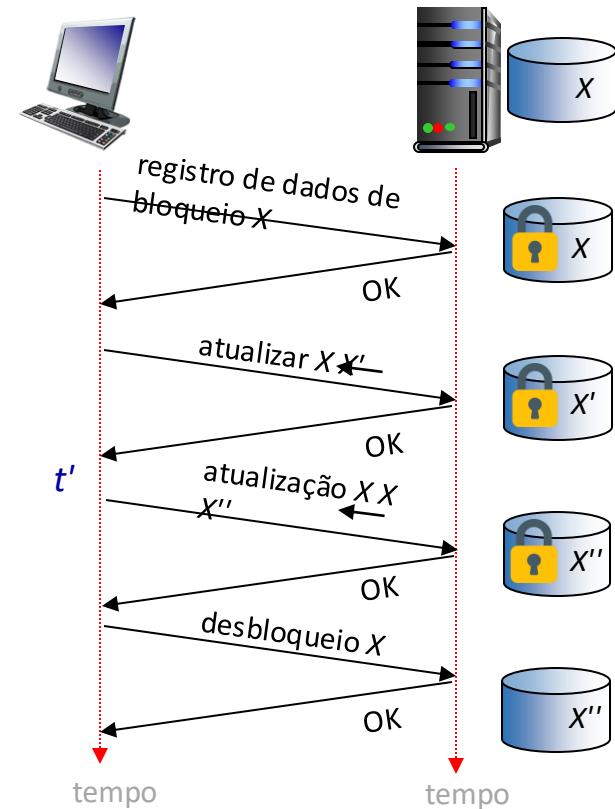
(ou use o Wireshark para examinar as solicitações/respostas HTTP capturadas)

Manutenção do estado do usuário/servidor: cookies

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

- nenhuma noção de trocas de mensagens HTTP em várias etapas para concluir uma "transação" na Web
 - não é necessário que o cliente/servidor rastreie o "estado" da troca em várias etapas
 - todas as solicitações HTTP são independentes umas das outras
 - não há necessidade de o cliente/servidor "recuperar-se" de uma transação parcialmente concluída, mas nunca completamente concluída

um protocolo com estado: o cliente faz duas alterações em X ou nenhuma



P: O que acontece se a conexão de rede ou o cliente falhar em t' ?

Manutenção do estado do usuário/servidor: cookies

Os sites da Web e o navegador do cliente usam *cookies* para manter algum estado entre as transações

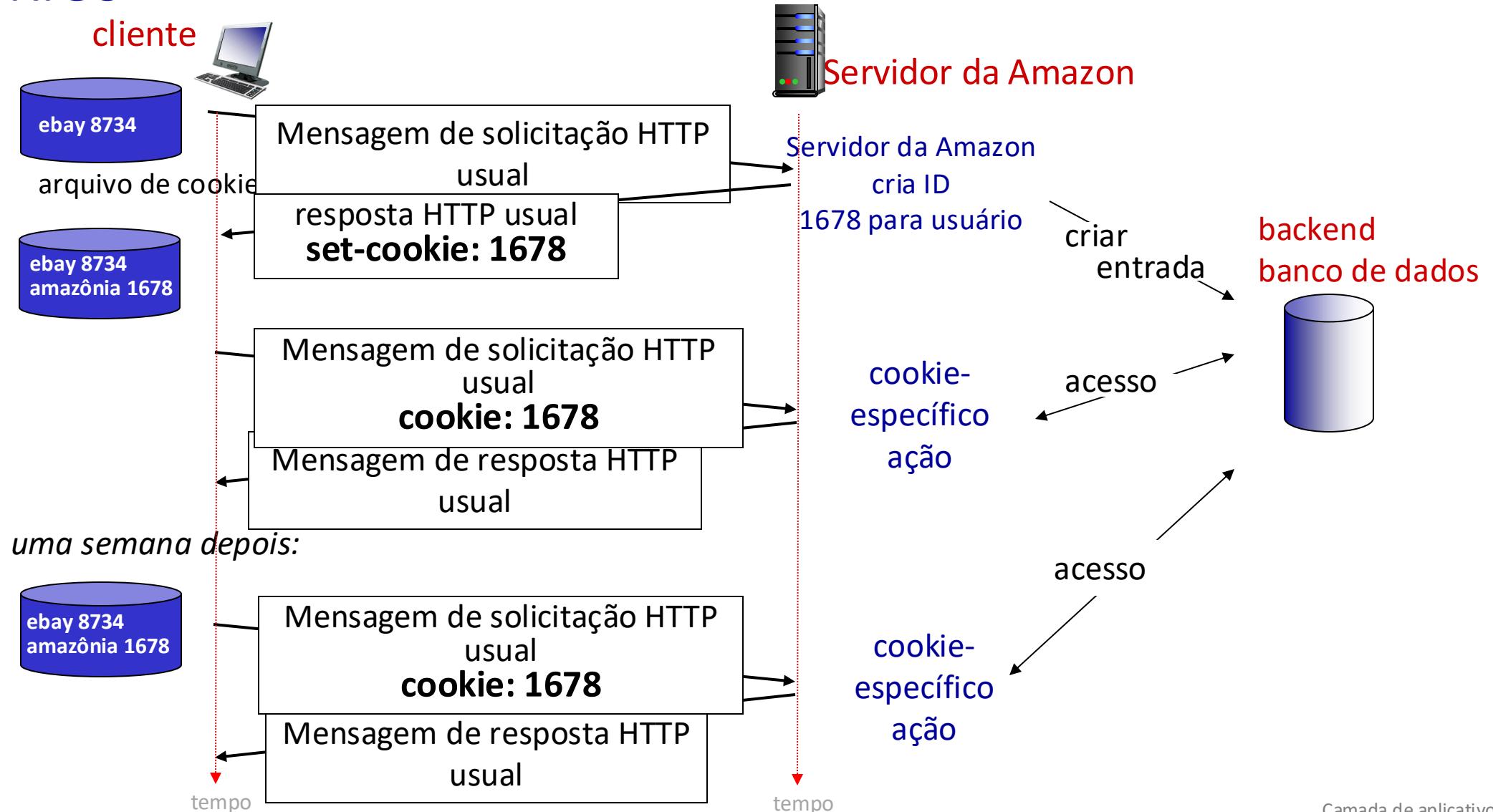
quatro componentes:

- 1) linha de cabeçalho de cookie da mensagem de *resposta* HTTP
- 2) linha do cabeçalho do 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 back-end no site

Exemplo:

- Susan usa o navegador no laptop e visita um site de comércio eletrônico específico pela primeira vez
- Quando as solicitações HTTP iniciais chegam ao site, o site cria:
 - ID exclusivo (também conhecido como "cookie")
 - no banco de dados backend para ID
 - As solicitações HTTP subsequentes de Susan para este site conterão o valor de ID do cookie, permitindo que o site "identifique" Susan

Manutenção do estado do usuário/servidor: cookies



Cookies HTTP: comentários

Para que os cookies podem ser usados:

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

Desafio: como manter o estado?

- *nos pontos de extremidade do protocolo:*
manter o estado no remetente/receptor em
várias transações
- *em mensagens:* os cookies em mensagens
HTTP carregam o estado

à parte

cookies e privacidade:

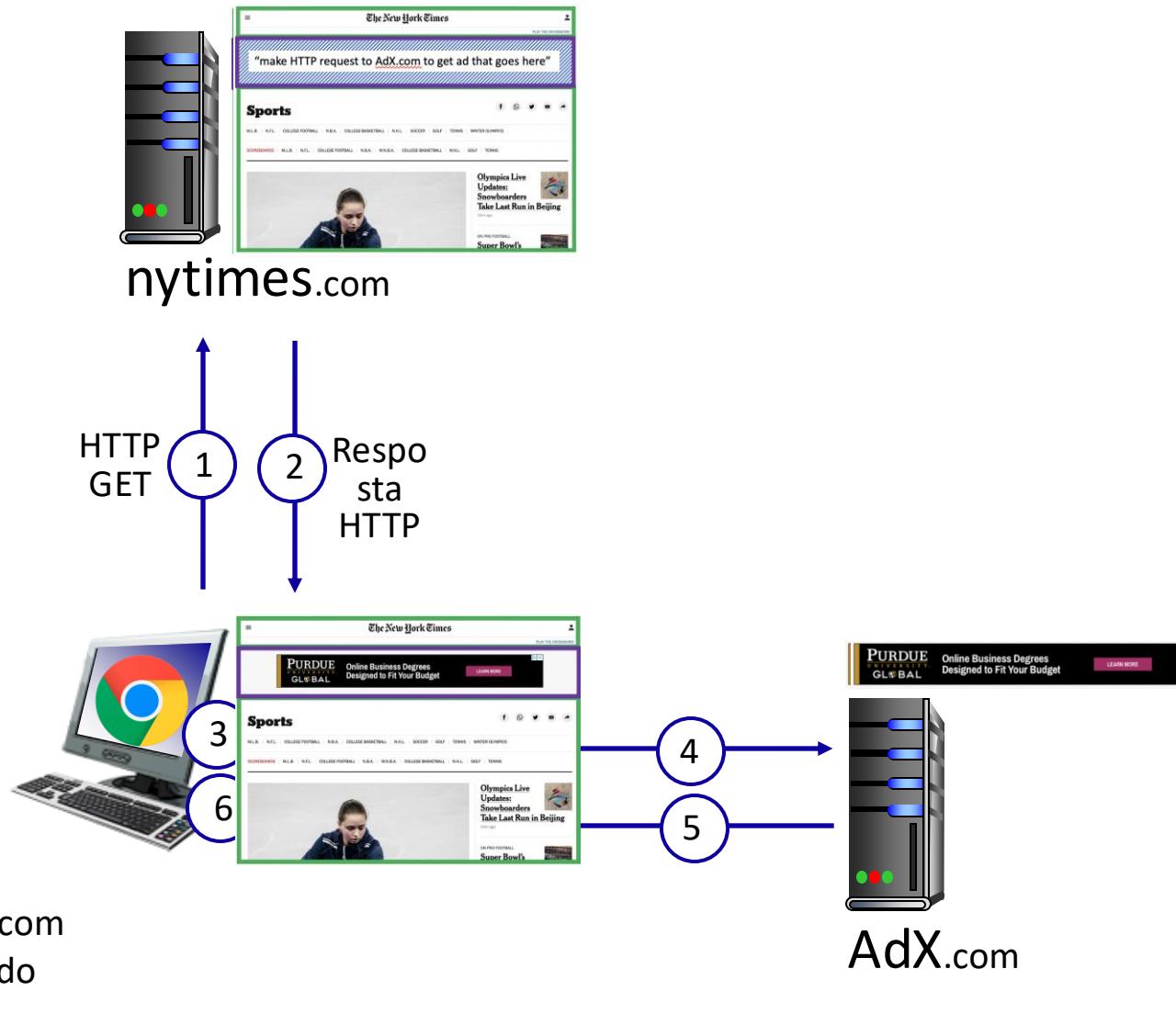
- Os cookies permitem que os sites *saibam* muito sobre você no site deles.
- cookies persistentes de terceiros (cookies de rastreamento) permitem que uma identidade comum (valor do cookie) seja rastreada em vários sites da Web

Exemplo: exibição de uma página da Web do NY Times

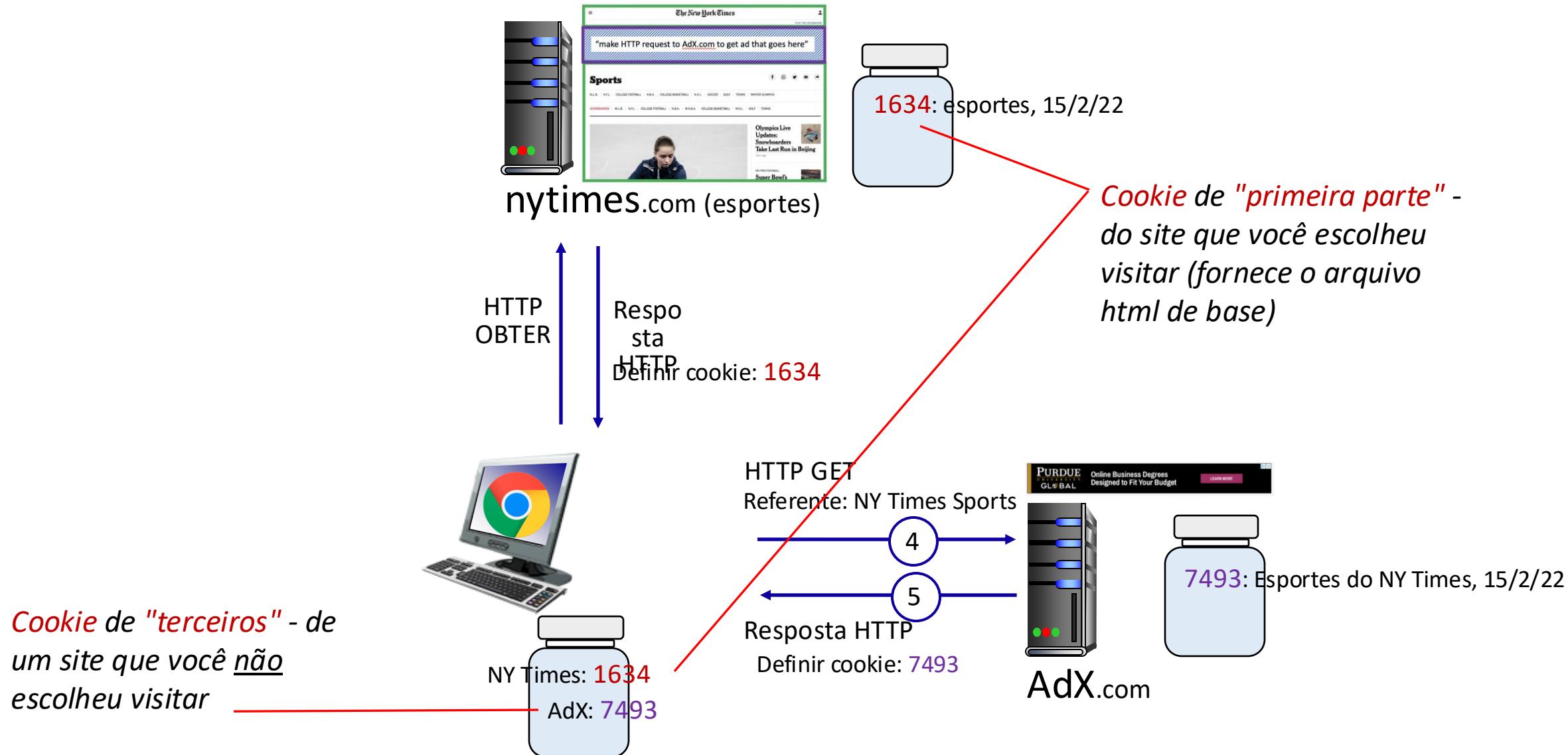
1 GET arquivo html básico
do site nytimes.com

4 buscar anúncio do
AdX.com

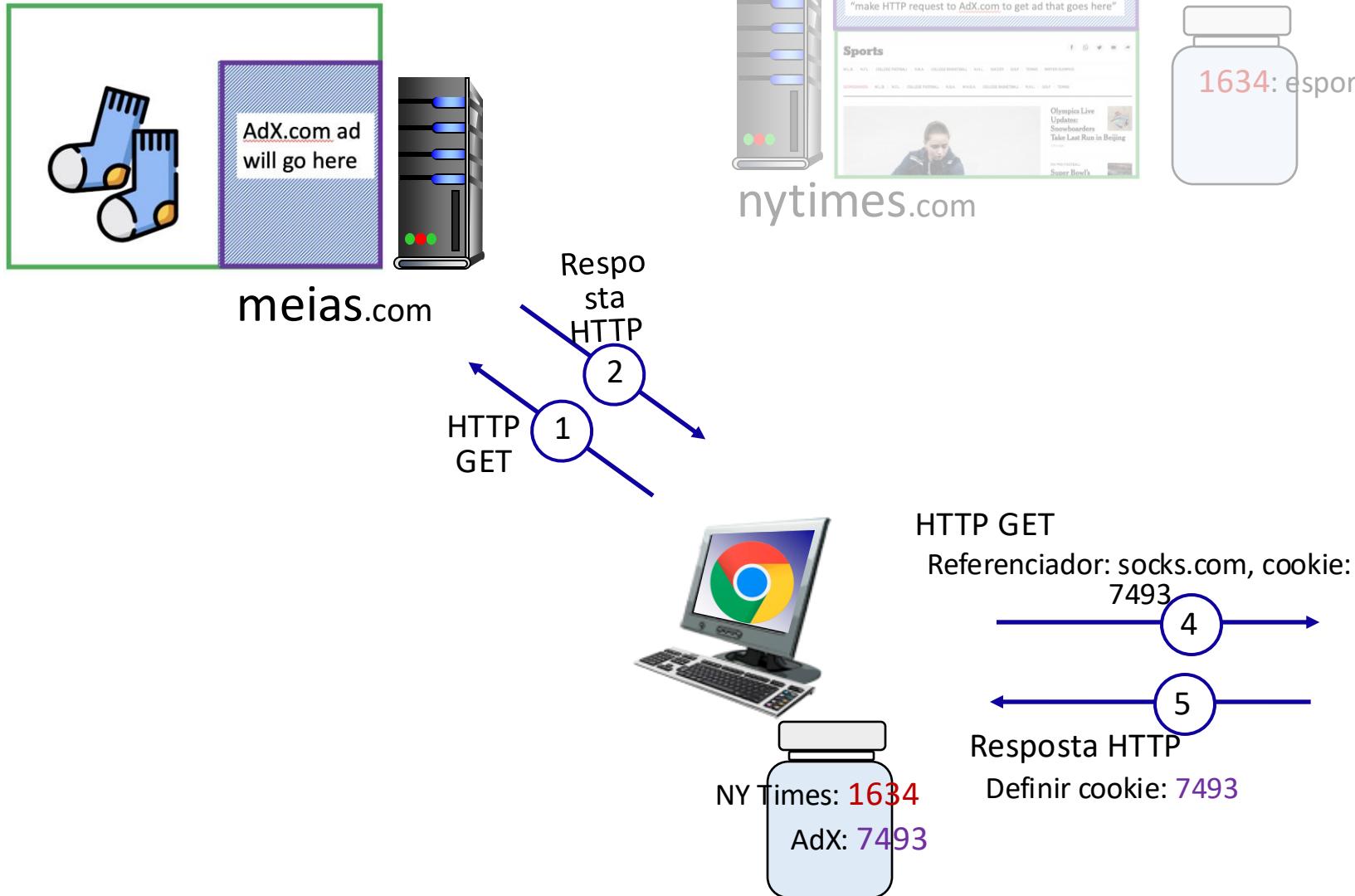
7 exibir página composta



Cookies: rastreamento do comportamento de navegação de um usuário



Cookies: rastreamento do comportamento de navegação de um usuário

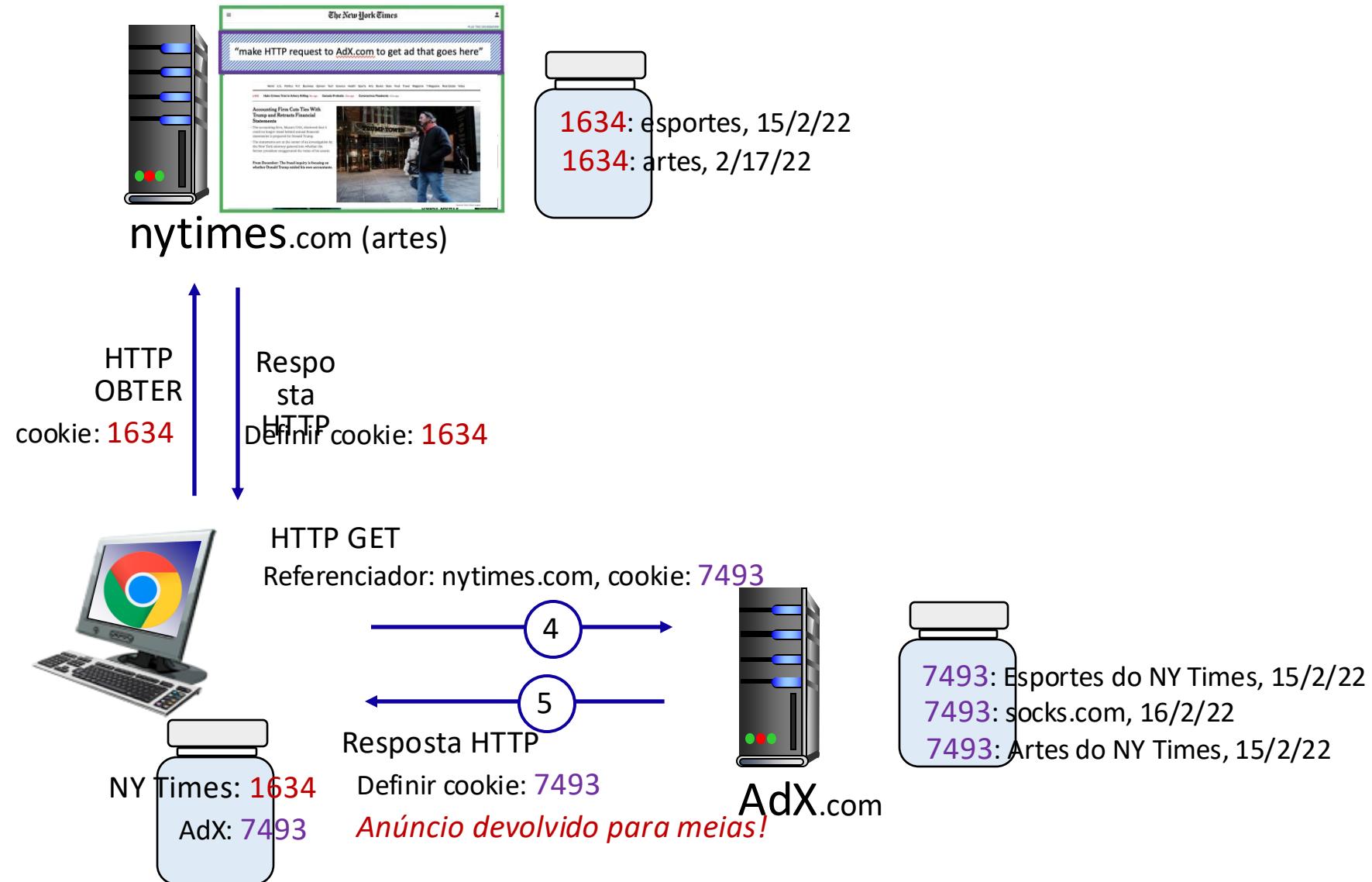


AdX:

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



Cookies: rastreiam o comportamento de navegação de um usuário (un



Cookies: rastrear o comportamento de navegação

Os cookies podem ser usados para:

- rastrear o comportamento do usuário em um determinado site (**cookies primários**)
- Rastrear o comportamento do usuário em vários sites (**cookies de terceiros**) sem que o usuário tenha escolhido visitar o site rastreador (!)
- O rastreamento pode ser *invisível* para o usuário:
 - em vez de o anúncio exibido acionar HTTP GET para o rastreador, poderia ser um link invisível

rastreamento de terceiros por meio de cookies:

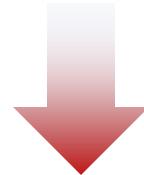
- desativado por padrão nos navegadores Firefox e Safari
- a ser desativado no navegador Chrome em 2023

GDPR (Regulamento Geral de Proteção de Dados da UE) e cookies

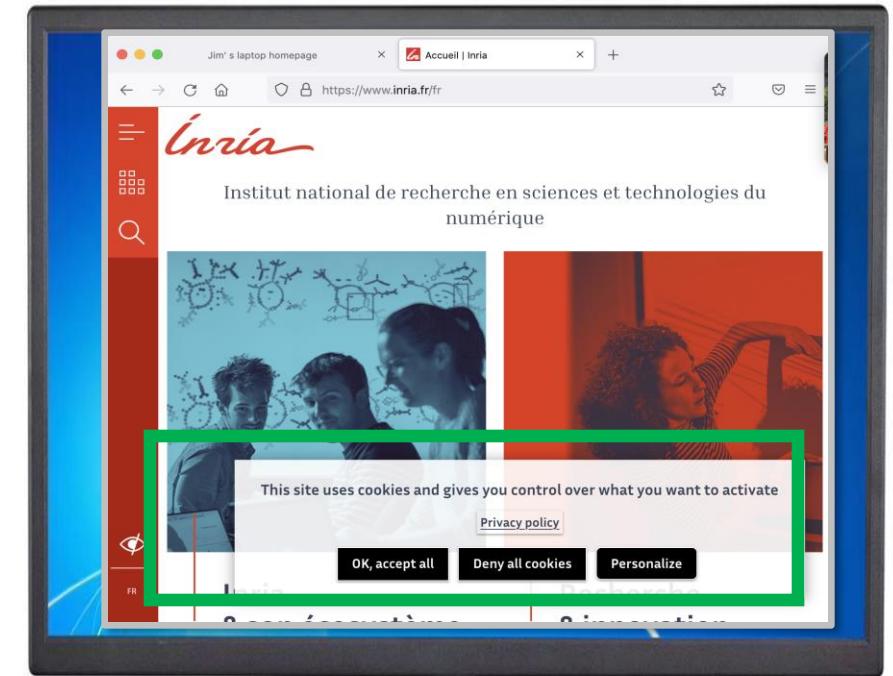
"As pessoas físicas podem ser associadas a identificadores on-line [...] como endereços de protocolo da Internet, identificadores de cookies ou outros identificadores [...].

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

GDPR, considerando 30 (maio de 2018)



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

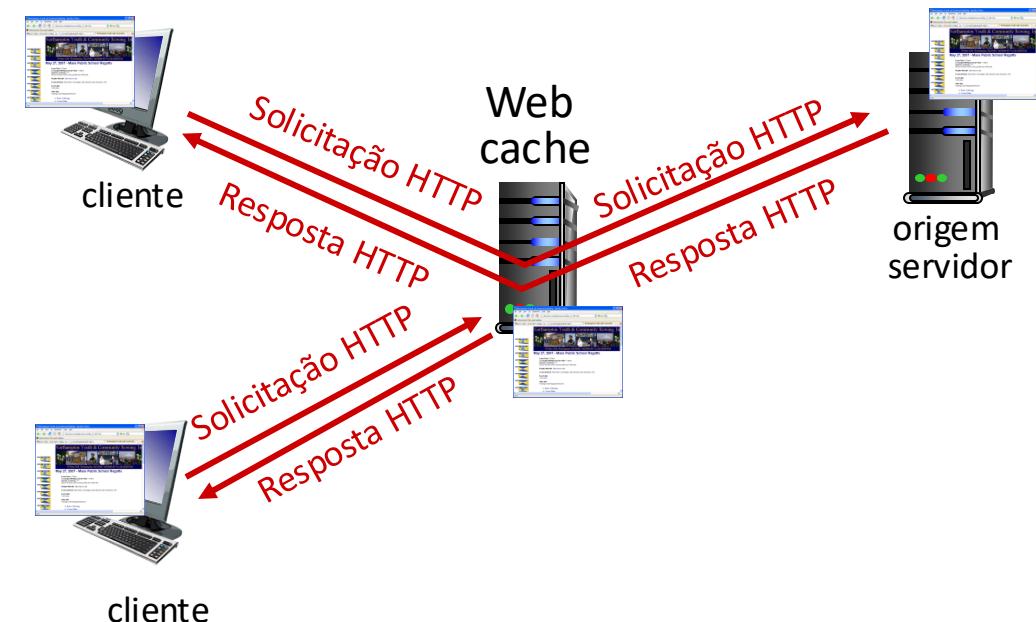


O usuário tem controle explícito sobre a permissão ou não de cookies

Caches da Web

Objetivo: atender às solicitações dos clientes sem envolver o servidor de origem

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



Caches da Web (também conhecidos como servidores proxy)

- O cache da Web atua como cliente e servidor
 - servidor para o cliente solicitante original
 - cliente para o servidor de origem
- o servidor informa ao cache sobre o cache permitido do objeto no cabeçalho da resposta:

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

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

Por que o cache da Web?

- reduzir o tempo de resposta da solicitação do cliente
 - o cache está mais próximo do cliente
- reduzir o tráfego no link de acesso de uma instituição
- A Internet está repleta de caches
 - permite que provedores de conteúdo "pobres" forneçam conteúdo de forma mais eficaz

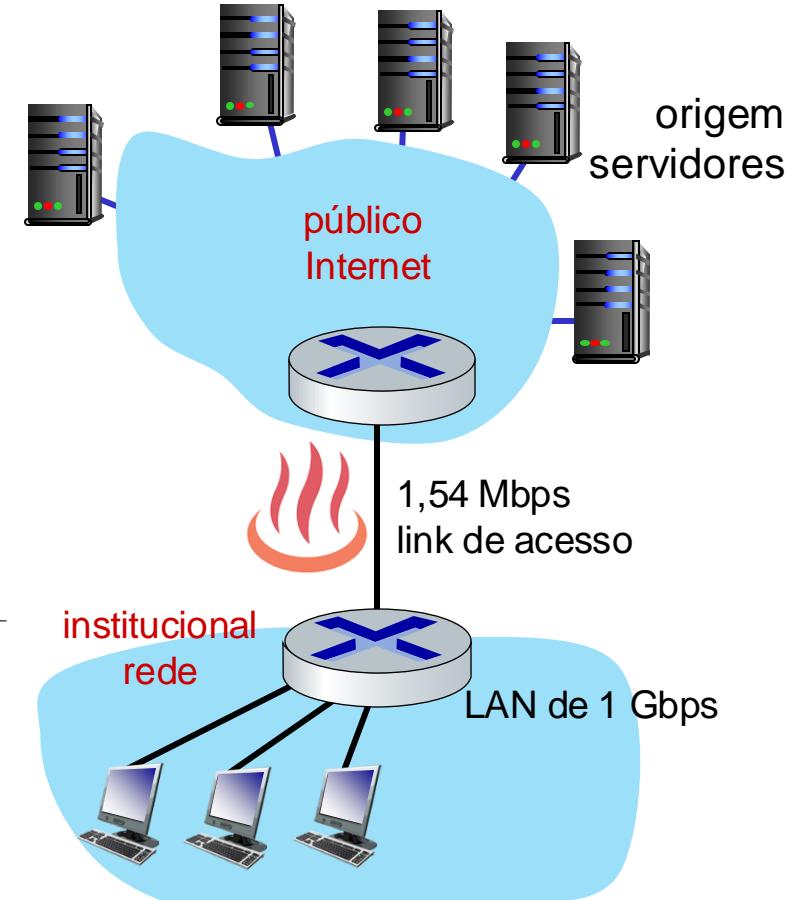
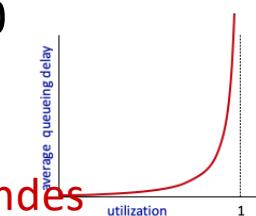
Exemplo de armazenamento em cache

Cenário:

- taxa de link de acesso: 1,54 Mbps
- RTT do roteador institucional para o servidor: 2 segundos
- Tamanho do objeto da Web: 100 mil bits
- taxa média de solicitação dos navegadores para os servidores de origem: 15/seg
 - Taxa média de dados para navegadores: 1,50 Mbps

Desempenho:

- utilização do link de acesso = 0,97
- Utilização da LAN: 0,0015
- atraso na extremidade = atraso na Internet + atraso do link de acesso + atraso da LAN
 - = 2 seg. + minutos + usecs



Opção 1: comprar um link de acesso mais rápido

Cenário:

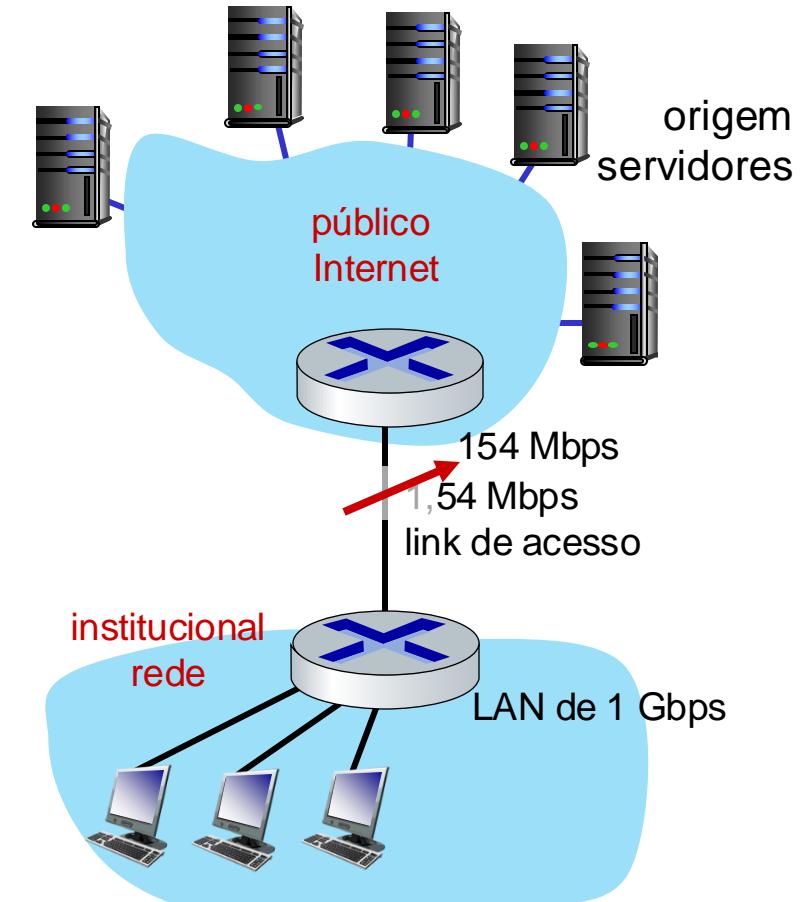
- taxa de link de acesso: 1,54 Mbps
- RTT do roteador institucional para o servidor: 2 segundos
- Tamanho do objeto da Web: 100 mil bits
- taxa média de solicitação dos navegadores para os servidores de origem: 15/seg
 - Taxa média de dados para navegadores: 1,50 Mbps

Desempenho:

- utilização do link de acesso = $0,97 \rightarrow .0097$
- Utilização da LAN: 0,0015
- atraso na extremidade = atraso na Internet + atraso do link de acesso + atraso da LAN

$$= 2 \text{ s} + \text{minutos} + \text{usecs}$$

Custo: link de acesso mais rápido (caro!)



Opção 2: instalar um cache da Web

Cenário:

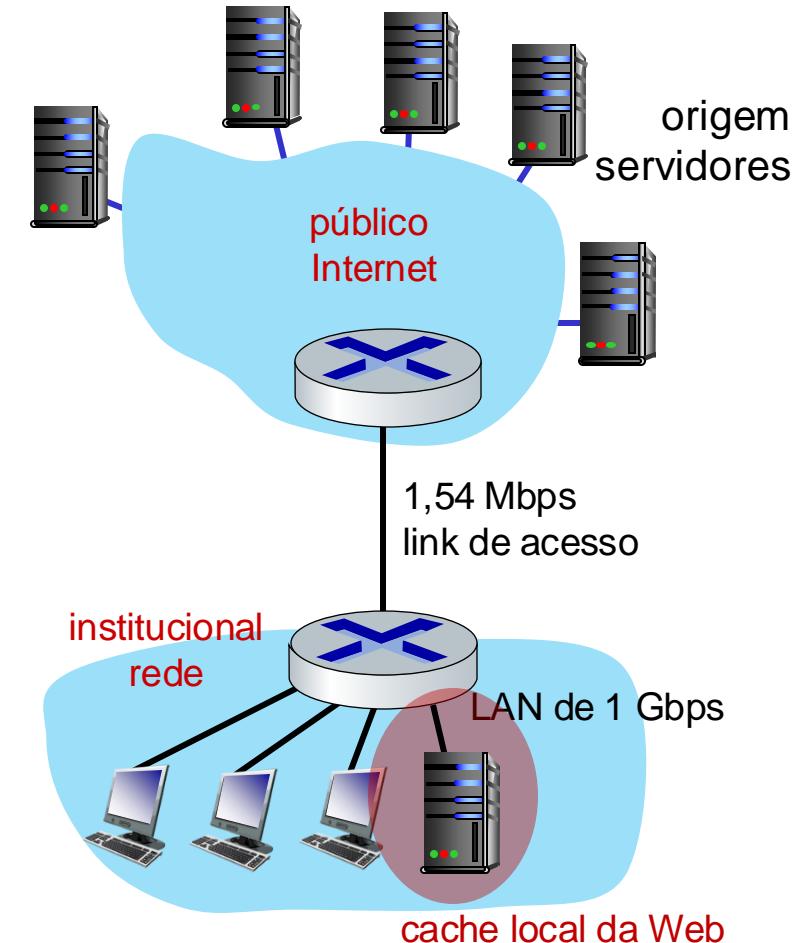
- taxa de link de acesso: 1,54 Mbps
- RTT do roteador institucional para o servidor: 2 segundos
- Tamanho do objeto da web: 100 mil bits
- taxa média de solicitação dos navegadores para os servidores de origem: 15/seg
 - Taxa média de dados para navegadores: 1,50 Mbps

Custo: cache da Web (barato!)

Desempenho:

- Utilização da LAN: ..?
- utilização do link de acesso = ?
- atraso médio final = ?

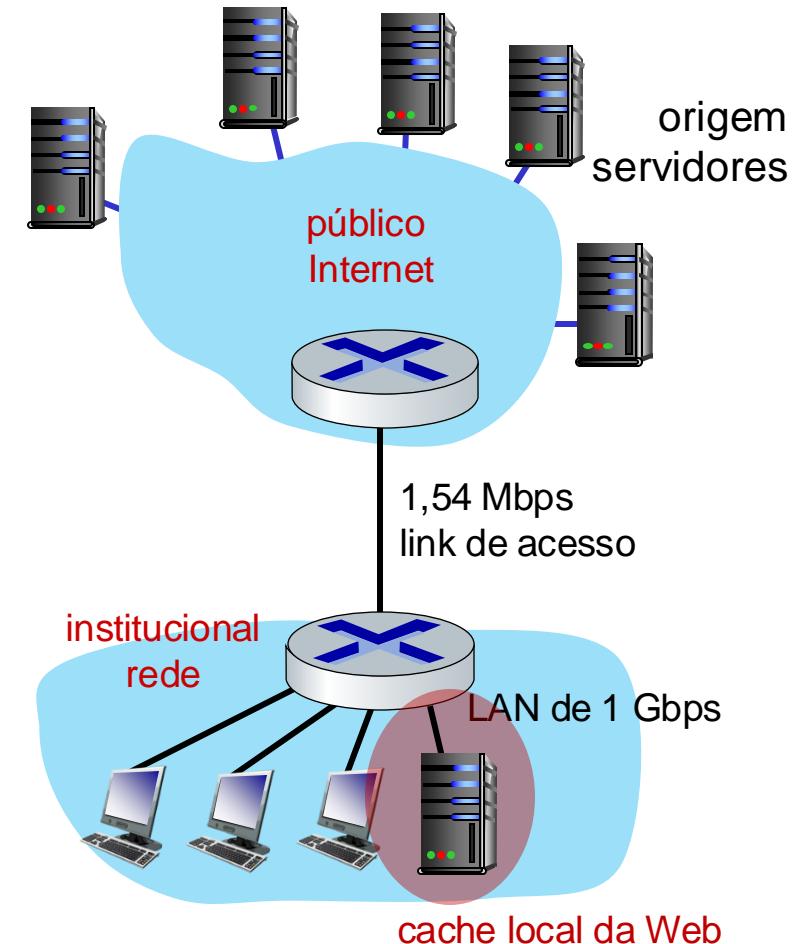
Como calcular o link utilização, atraso?



Cálculo da utilização do link de acesso, atraso final com cache:

suponha que a taxa de acerto do cache seja 0,4:

- 40% das solicitações atendidas pelo cache, com baixo atraso (mseg)
- 60% das solicitações atendidas na origem
 - taxa para navegadores pelo link de acesso
 $= 0,6 * 1,50 \text{ Mbps} = 0,9 \text{ Mbps}$
 - utilização do link de acesso $= 0,9 / 1,54 = 0,58$ significa baixo atraso de enfileiramento (mseg) no link de acesso
- atraso médio final:
 $= 0,6 * (\text{atraso dos servidores de origem}) + 0,4 * (\text{atraso quando satisfeito no cache})$
 $= 0,6 (2,01) + 0,4 (\sim \text{msecs}) = \sim 1,2 \text{ segs}$

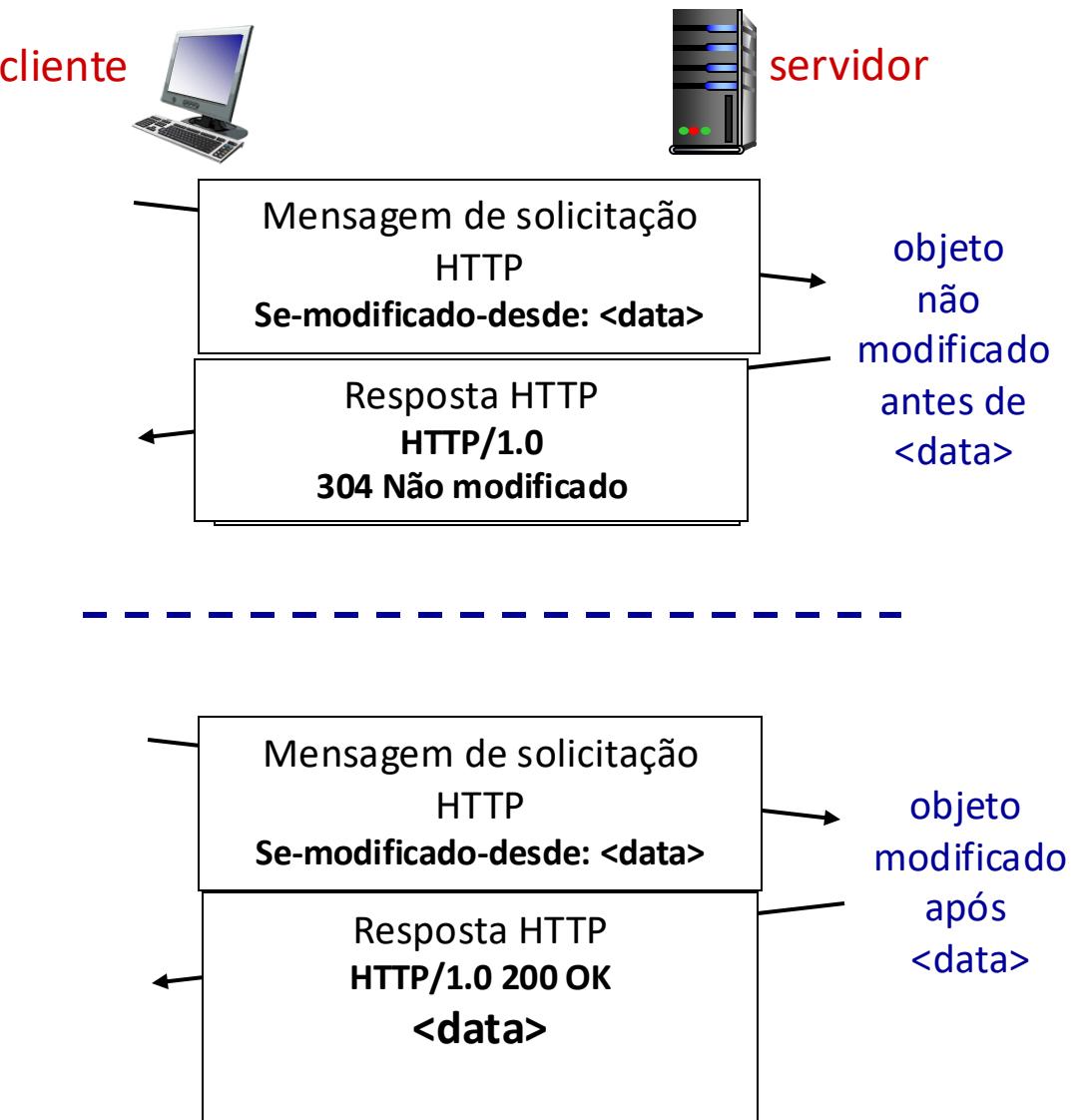


atraso médio final menor do que com um link de 154 Mbps (e mais barato também!)

Cache do navegador: GET condicional

Objetivo: não enviar o objeto se o navegador tiver uma versão atualizada em cache

- sem atraso na transmissão de objetos (ou uso de recursos de rede)
- **cliente:** especifica a data da cópia armazenada em cache pelo navegador na solicitação HTTP
Se-modificado-desde: <data>
- **servidor:** a resposta não contém nenhum objeto se a cópia armazenada em cache pelo navegador estiver atualizada:
HTTP/1.0 304 Não modificado



HTTP/2

Objetivo principal: redução do atraso em solicitações
HTTP de vários objetos

HTTP1.1: introduziu vários GETs em pipeline em uma única conexão TCP

- o servidor responde *em ordem* (FCFS: agendamento por ordem de chegada) às solicitações GET
- Com o FCFS, o objeto pequeno pode ter que esperar pela transmissão (**bloqueio de head-of-line (HOL)**) atrás de objetos grandes
- a recuperação de perdas (retransmissão de segmentos TCP perdidos) interrompe a transmissão de objetos

HTTP/2

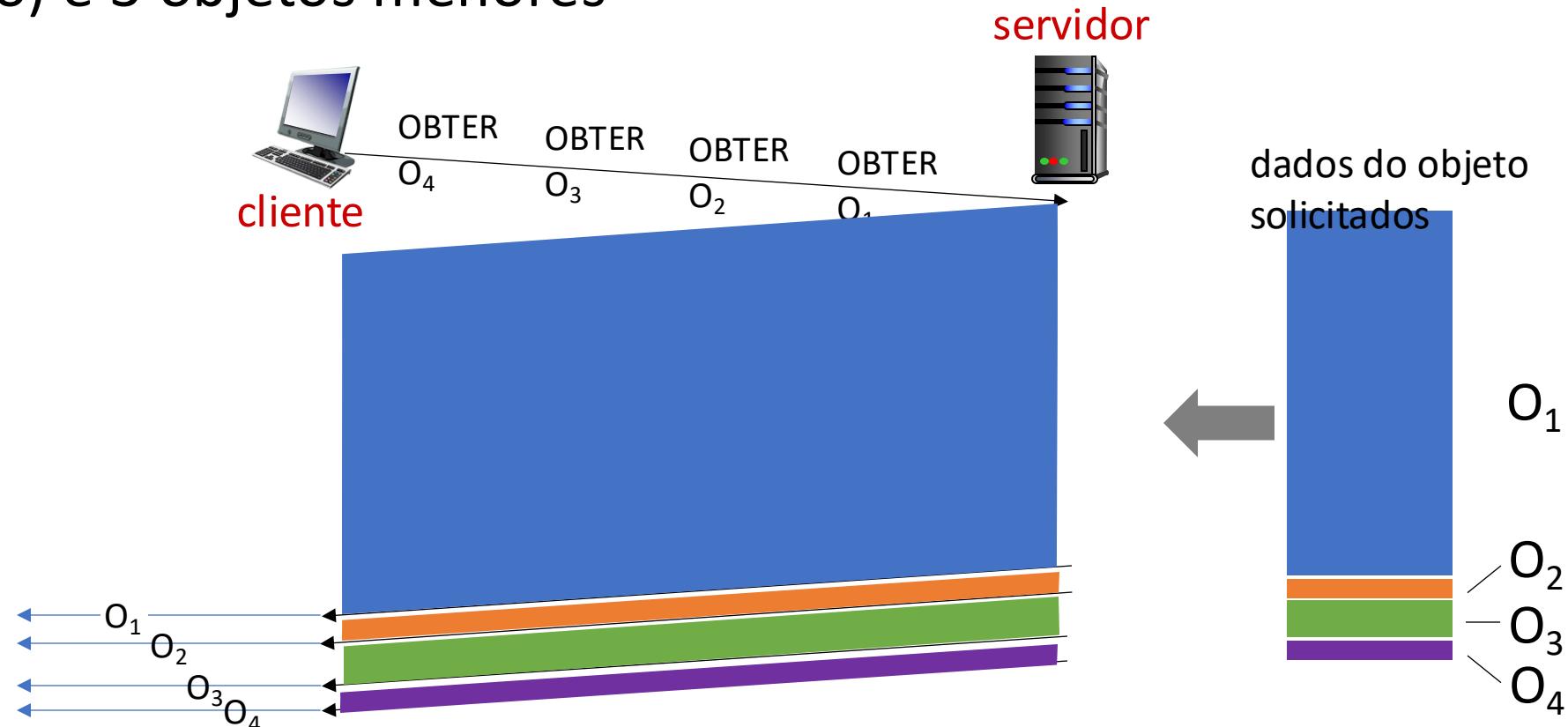
Objetivo principal: redução do atraso em solicitações HTTP de vários objetos

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

- métodos, códigos de status, a maioria dos campos de cabeçalho inalterados em relação ao HTTP 1.1
- ordem de transmissão dos objetos solicitados com base na prioridade de objeto especificada pelo cliente (não necessariamente FCFS)
- *enviar* objetos não solicitados para o cliente
- dividir os objetos em quadros, programar quadros para reduzir o bloqueio do HOL

HTTP/2: mitigando o bloqueio do HOL

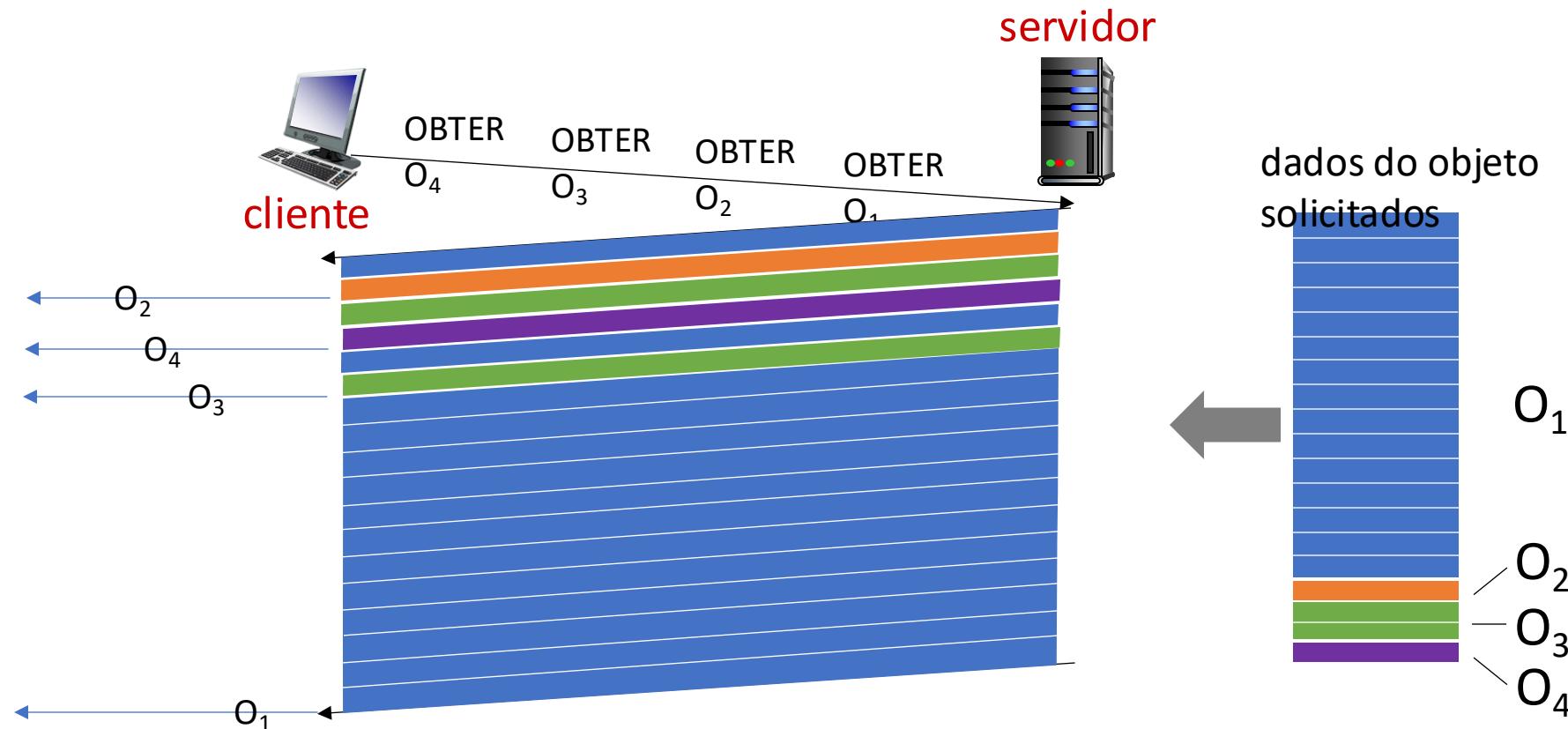
HTTP 1.1: o cliente solicita 1 objeto grande (por exemplo, arquivo de vídeo) e 3 objetos menores



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

HTTP/2: mitigando o bloqueio do HOL

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



O_2, O_3, O_4 entregue rapidamente, O_1 ligeiramente atrasado

HTTP/2 para HTTP/3

HTTP/2 em uma única conexão TCP significa:

- a recuperação da perda de pacotes ainda interrompe todas as transmissões de objetos
 - Como no HTTP 1.1, os navegadores são incentivados a abrir várias conexões TCP paralelas para reduzir a paralisação e aumentar a taxa de transferência geral
- nenhuma segurança sobre a conexão TCP básica
- **HTTP/3**: adiciona segurança, controle de erros e congestionamento por objeto (mais pipelining) sobre UDP
 - mais sobre HTTP/3 na camada de transporte

Camada de aplicativos: visão geral

- Princípios de aplicativos de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O sistema de nomes de domínio DNS
- Aplicativos P2P
- redes de distribuição de conteúdo e streaming de vídeo
- programação de soquetes com UDP e TCP



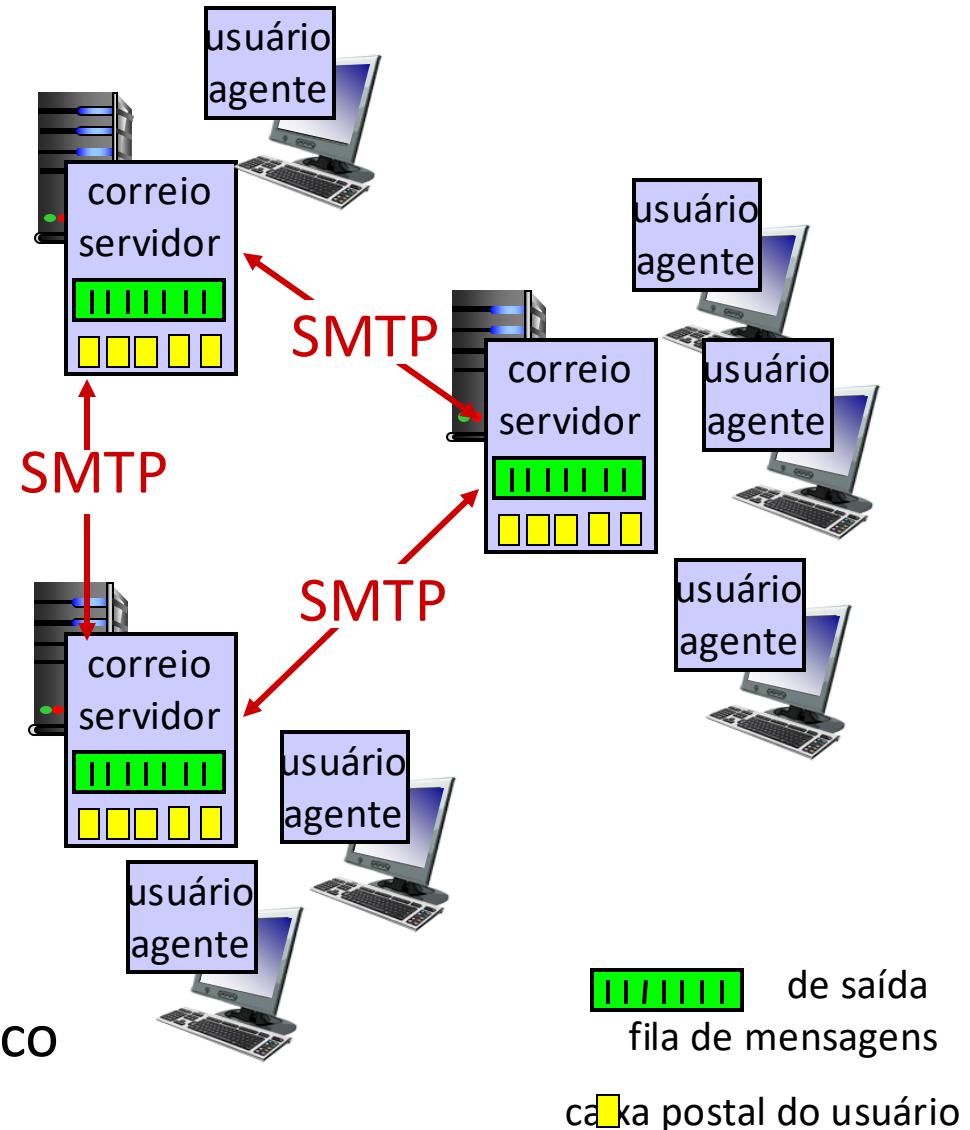
E-mail

Três componentes principais:

- agentes de usuários
- servidores de correio eletrônico
- protocolo simples de transferência de correio eletrônico: SMTP

Agente do usuário

- também conhecido como "leitor de correio eletrônico"
- composição, edição e leitura de mensagens de correio eletrônico
- Por exemplo, Outlook, cliente de correio eletrônico do iPhone
- mensagens enviadas e recebidas armazenadas no servidor



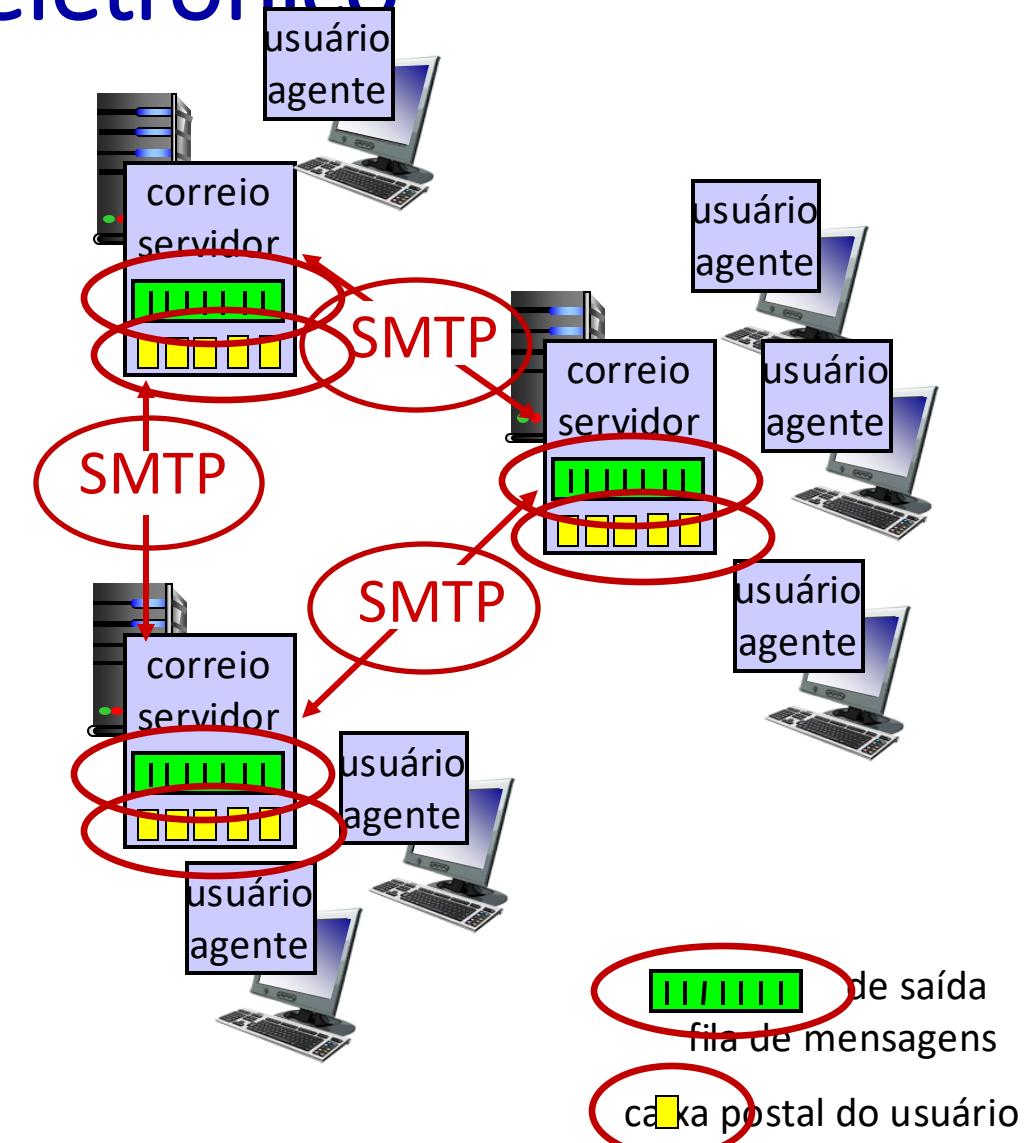
E-mail: servidores de correio eletrônico

servidores de correio eletrônico:

- *A caixa postal* contém mensagens recebidas para o usuário
- *fila de mensagens* de correio eletrônico de saída (a serem enviadas)

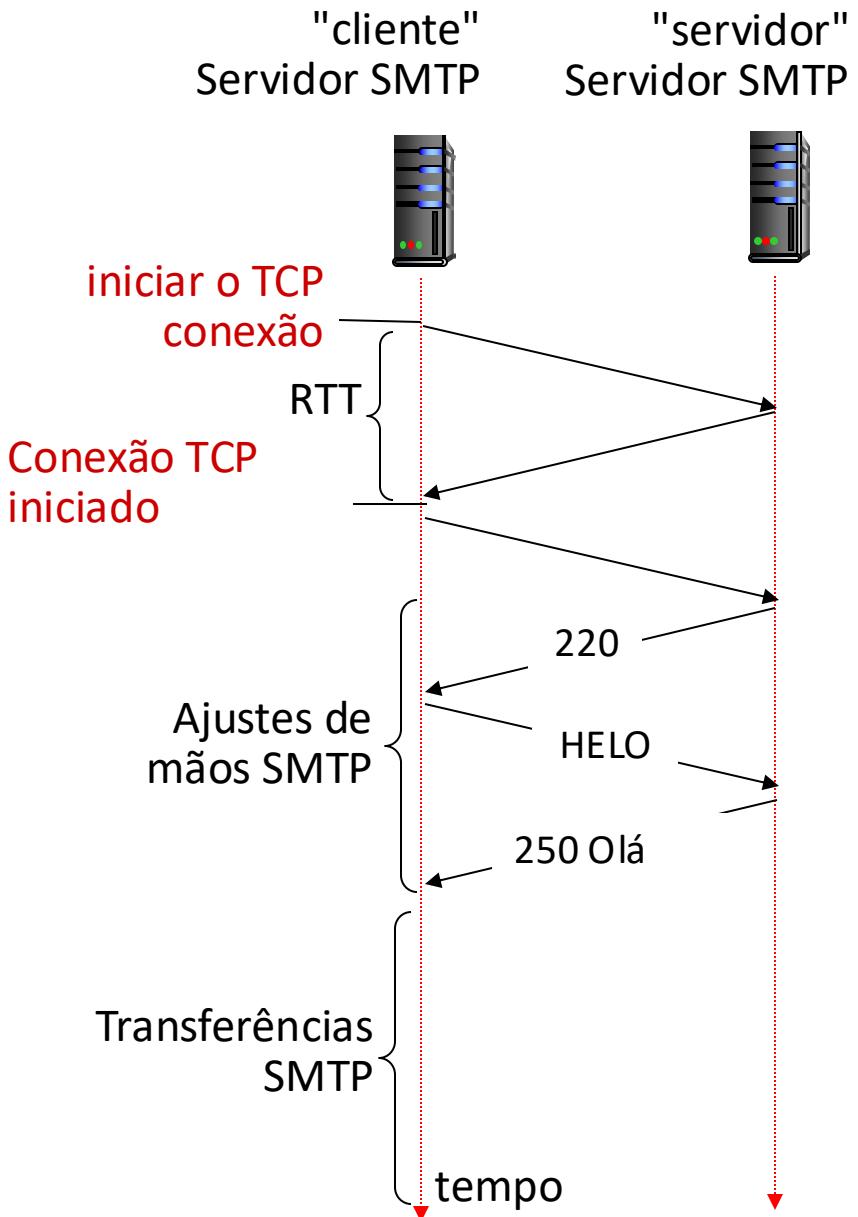
Protocolo **SMTP** entre servidores de correio eletrônico para enviar mensagens de e-mail

- *cliente*: servidor de correio eletrônico de envio
- "*server*": servidor de recebimento de correio eletrônico



SMTP RFC (5321)

- usa TCP para transferir de forma confiável a mensagem de e-mail do cliente (servidor de e-mail que inicia a conexão) para o servidor, porta 25
 - transferência direta: servidor de envio (agindo como cliente) para o servidor de recebimento
- três fases de transferência
 - Aperto de mão SMTP (saudação)
 - Transferência de mensagens por SMTP
 - Fechamento de SMTP
- interação de comando/resposta (como HTTP)
 - comandos: Texto ASCII
 - resposta: código de status e frase



Cenário: Alice envia um e-mail para Bob

1) Alice usa o UA para redigir uma mensagem de e-mail "para" bob@someschool.edu

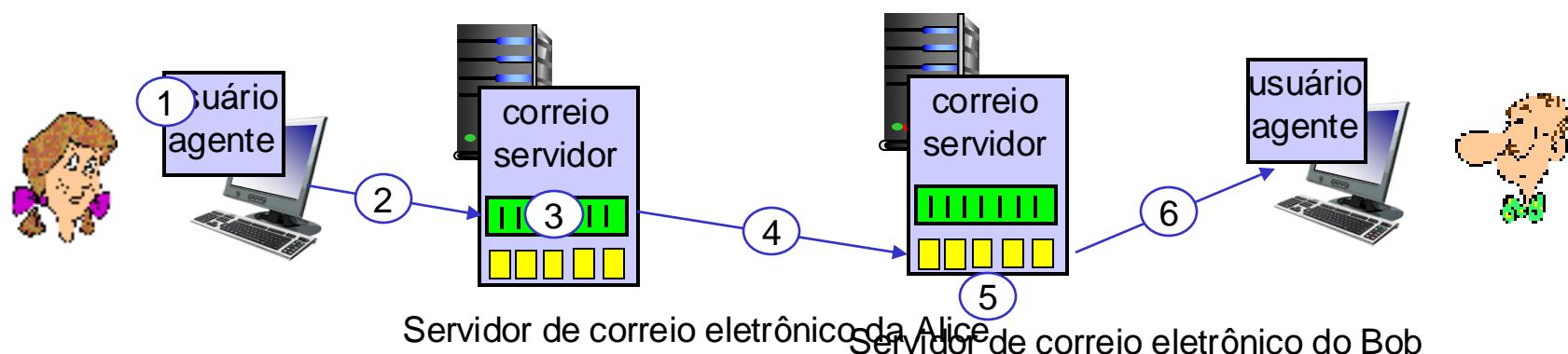
2) O UA de Alice envia uma mensagem para seu servidor de e-mail usando SMTP; a mensagem é colocada na fila de mensagens

3) O lado cliente do SMTP no servidor de correio eletrônico abre uma conexão TCP com o servidor de correio eletrônico de Bob

4) O cliente SMTP envia a mensagem de Alice pela conexão TCP

5) O servidor de correio eletrônico de Bob coloca a mensagem na caixa postal de Bob

6) Bob invoca seu agente de usuário para ler a mensagem



Exemplo de interação SMTP

S: 220 hamburger.edu

separad

SMTP: observações

comparação com HTTP:

- HTTP: pull do cliente
- SMTP: envio do cliente
- Ambos têm interação de comando/resposta ASCII, códigos de status
- HTTP: cada objeto encapsulado em sua própria mensagem de resposta
- SMTP: vários objetos enviados em uma mensagem de várias partes

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

Formato da mensagem de correio eletrônico

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

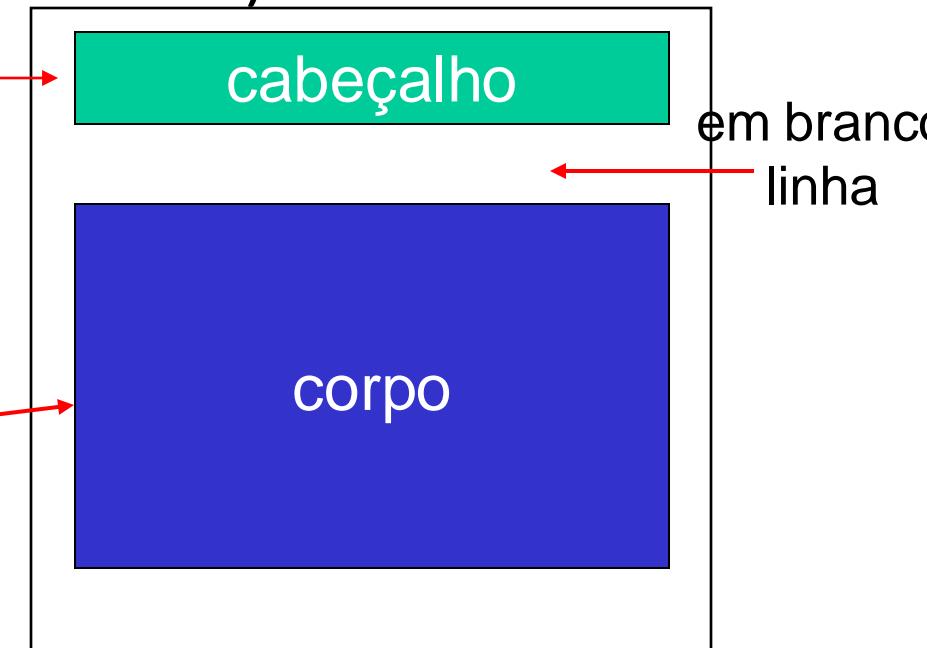
A RFC 2822 define a *sintaxe* da própria mensagem de e-mail (assim como o HTML define a sintaxe dos documentos da Web)

- linhas de cabeçalho, por exemplo,

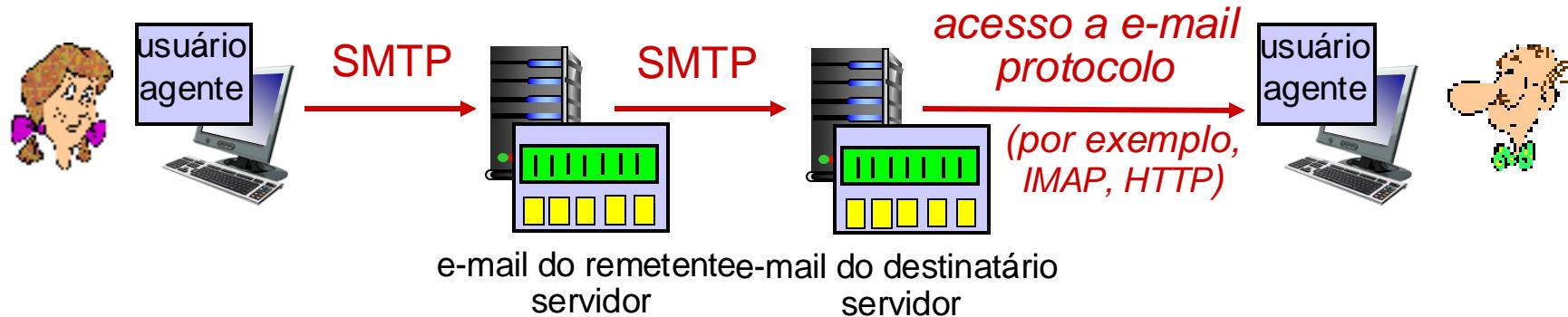
- Para:
- De:
- Assunto:

Essas linhas, no corpo da área da mensagem de e-mail, são diferentes dos comandos SMTP
~~MAIL FROM:, RCPT TO:!~~

- Corpo: a "mensagem", somente caracteres ASCII



Recuperação de e-mail: protocolos de acesso a correio eletrônico



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

Camada de aplicativos: Visão geral

- Princípios de aplicativos de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O sistema de nomes de domínio DNS
- Aplicativos P2P
- redes de distribuição de conteúdo e streaming de vídeo
- programação de soquetes com UDP e TCP



DNS: Sistema de Nomes de Domínios

pessoas: muitos identificadores:

- SSN, nome, número do passaporte

Hosts da Internet, roteadores:

- Endereço IP (32 bits) - usado para endereçar datagramas
- "name" (nome), por exemplo, cs.umass.edu - usado por humanos

P: como mapear entre o endereço IP e o nome, e vice-versa?

Sistema de Nomes de Domínio (DNS):

- *banco de dados distribuído* implementado na hierarquia de muitos *servidores de nomes*
- *protocolo de camada de aplicativo:* hosts, servidores DNS se comunicam para *resolver* nomes (tradução de endereço/nome)
 - *Observação:* função central da Internet, implementada como protocolo de camada de aplicativo
 - complexidade na "borda" da rede

DNS: serviços, estrutura

Serviços de DNS:

- tradução de nome de host para endereço IP
- aliasing de host
 - canônico, nomes de alias
- aliasing de servidor de correio eletrônico
- distribuição de carga
 - servidores da Web replicados: muitos endereços IP correspondem a um nome

P: Por que não centralizar o DNS?

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

R: não é escalonável!

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

Pensando no DNS

um enorme banco de dados distribuído:

- ~ bilhões de registros, cada um simples

lida com muitos *trilhões* de consultas/dia:

- *muito* mais leituras do que gravações
- *O desempenho é importante*: quase todas as transações da Internet interagem com o DNS - os msecs contam!

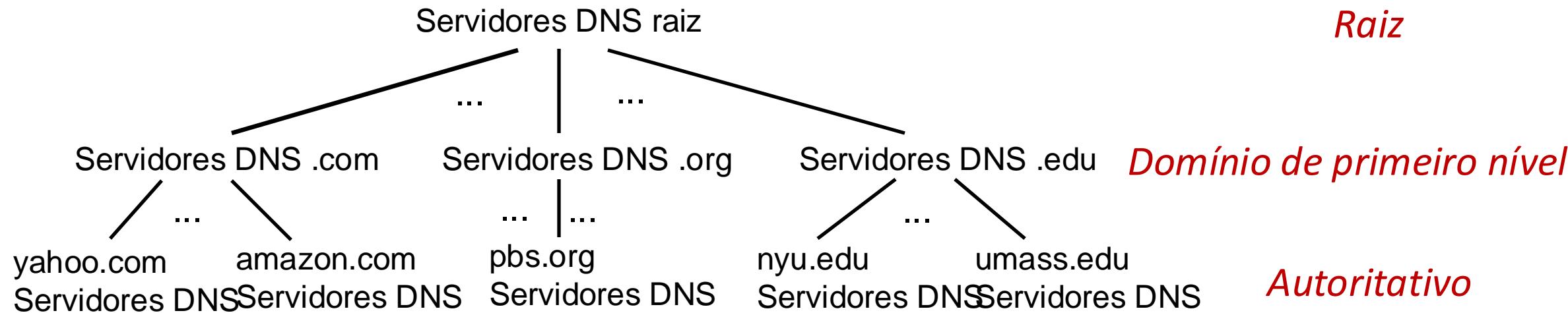
descentralizado em termos organizacionais e físicos:

- milhões de organizações diferentes responsáveis por seus registros

"À prova de balas": confiabilidade, segurança



DNS: um banco de dados distribuído e hierárquico

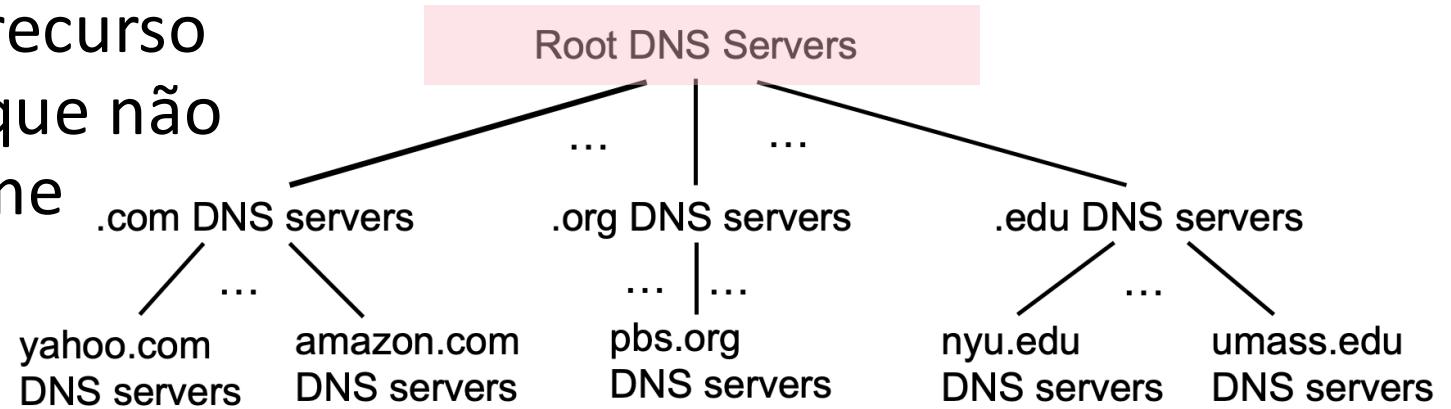


O cliente deseja um endereço IP para www.amazon.com; 1 aproximação dest :

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

DNS: servidores de nomes raiz

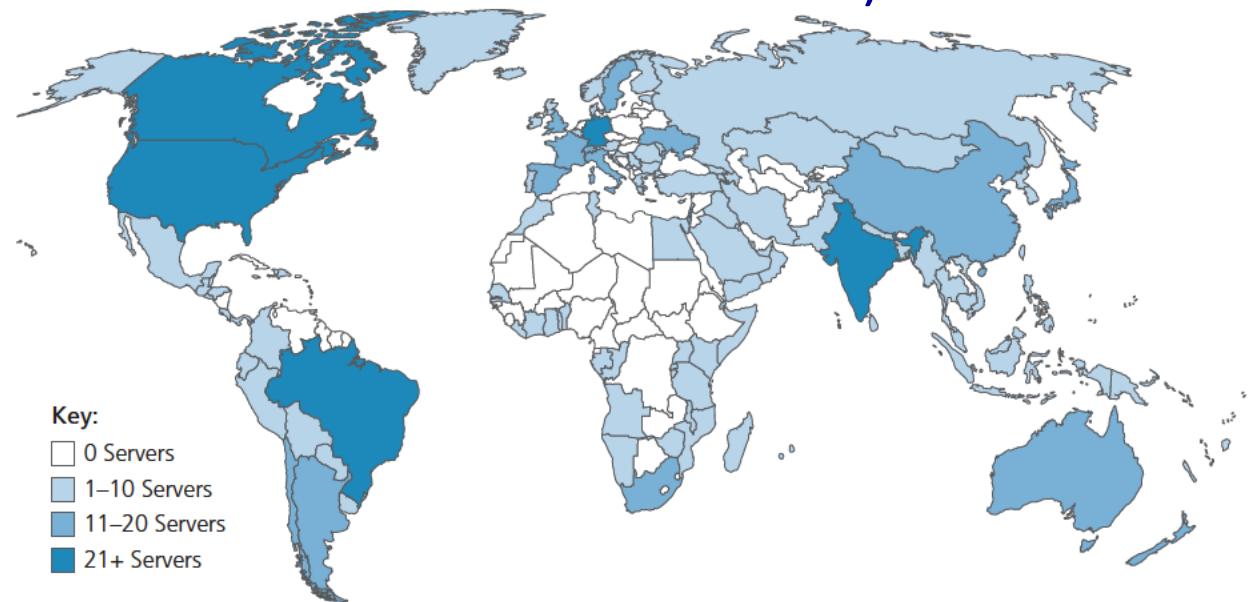
- oficial, contato de último recurso por servidores de nomes que não conseguem resolver o nome



DNS: servidores de nomes raiz

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

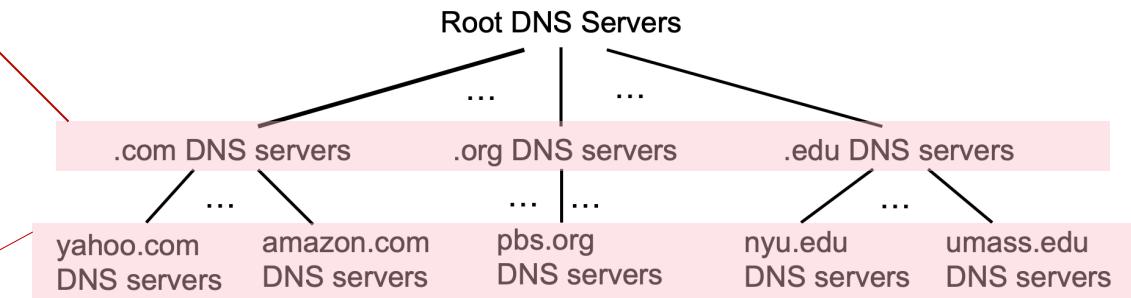
13 "servidores" de nomes de raiz lógica em todo o mundo, cada "servidor" replicado várias vezes (cerca de 200 servidores nos EUA)



Domínio de nível superior e servidores autoritativos

Servidores de TLD (Top-Level Domain):

- responsável por .com, .org, .net, .edu, .aero, .jobs, .museums e todos os domínios de país de nível superior, por exemplo: .cn, .uk, .fr, .ca, .jp
- Network Solutions: registro autorizado para TLDs .com e .net
- Educause: TLD .edu



servidores DNS autoritativos:

- servidor(es) DNS próprio(s) da organização, fornecendo mapeamentos autoritativos de nome de host para IP para os hosts nomeados da organização
- pode ser mantido pela organização ou pelo provedor de serviços

Servidores de nomes DNS locais

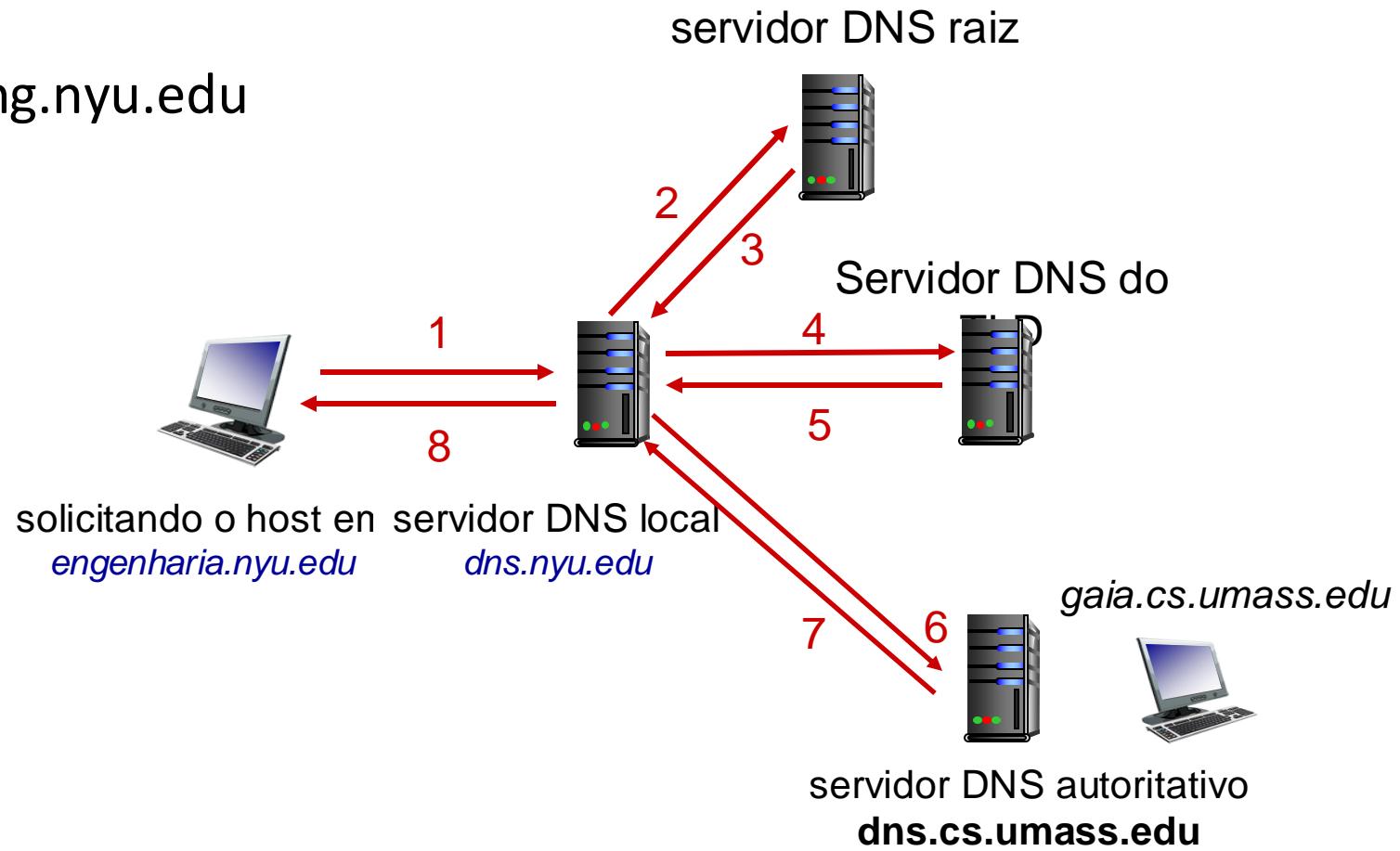
- Quando o host faz uma consulta de DNS, ela é enviada ao seu servidor DNS *local*
 - O servidor DNS local retorna a resposta, respondendo:
 - de seu cache local de pares de tradução de nome para endereço recentes (possivelmente desatualizados!)
 - encaminhamento da solicitação para a hierarquia do DNS para resolução
 - Cada ISP tem um servidor de nomes DNS local; para encontrar o seu:
 - MacOS: % scutil --dns
 - Windows: >ipconfig /all
- o servidor DNS local não pertence estritamente à hierarquia

Resolução de nomes DNS: consulta iterada

Exemplo: host em engineering.nyu.edu deseja o endereço IP de gaia.cs.umass.edu

Consulta iterada:

- O servidor contatado responde com o nome do servidor a ser contatado
- "Eu não sei esse nome, mas pergunte a este servidor"

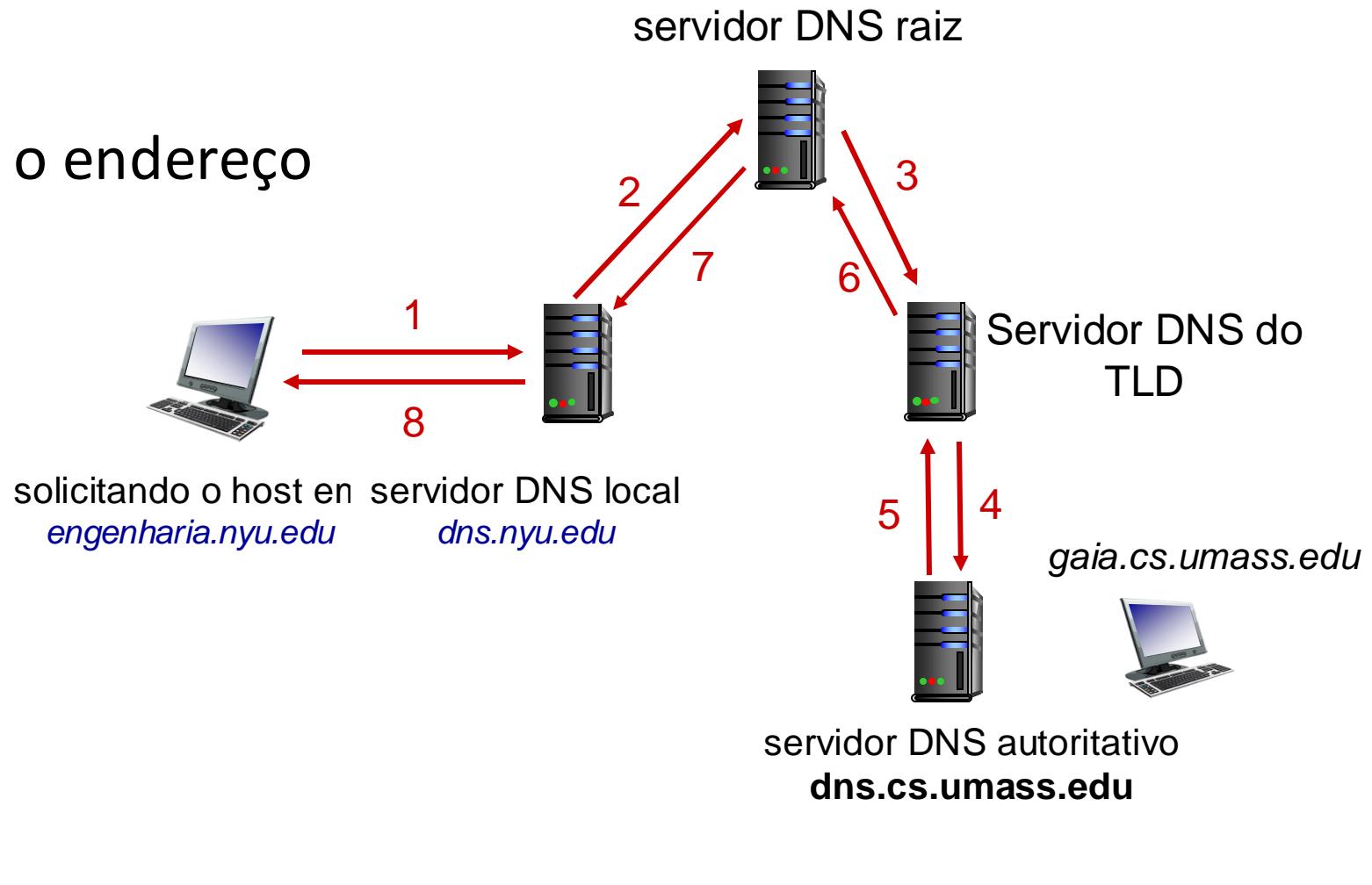


Resolução de nomes DNS: consulta recursiva

Exemplo: o host em engineering.nyu.edu deseja o endereço IP de gaia.cs.umass.edu

Consulta recursiva:

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



Armazenamento em cache de informações de DNS

- Quando (qualquer) servidor de nomes aprende o mapeamento, ele *armazena* o mapeamento *em cache* e retorna *imediatamente* um mapeamento em cache em resposta a uma consulta
 - O armazenamento em cache melhora o tempo de resposta
 - as entradas do cache expiram (desaparecem) após algum tempo (TTL)
 - Servidores TLD normalmente armazenados em cache nos servidores de nomes locais
- as entradas armazenadas em cache podem estar *desatualizadas*
 - Se o host nomeado mudar de endereço IP, ele poderá não ser conhecido em toda a Internet até que todos os TTLs expirem!
 - *Tradução de nome para endereço com o melhor esforço possível!*

Registros DNS

DNS: banco de dados distribuído que armazena registros de recursos (**RR**)

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

type=A

- nome é hostname
- valor é o endereço IP

type=NS

- o nome é o domínio (por exemplo, foo.com)
- valor é o nome do host do servidor de nomes autoritativo para esse domínio

type=CNAME

- nome é um nome de alias para algum nome "canônico" (o verdadeiro)
- www.ibm.com é realmente servereast.backup2.ibm.com
- valor é o nome canônico

type=MX

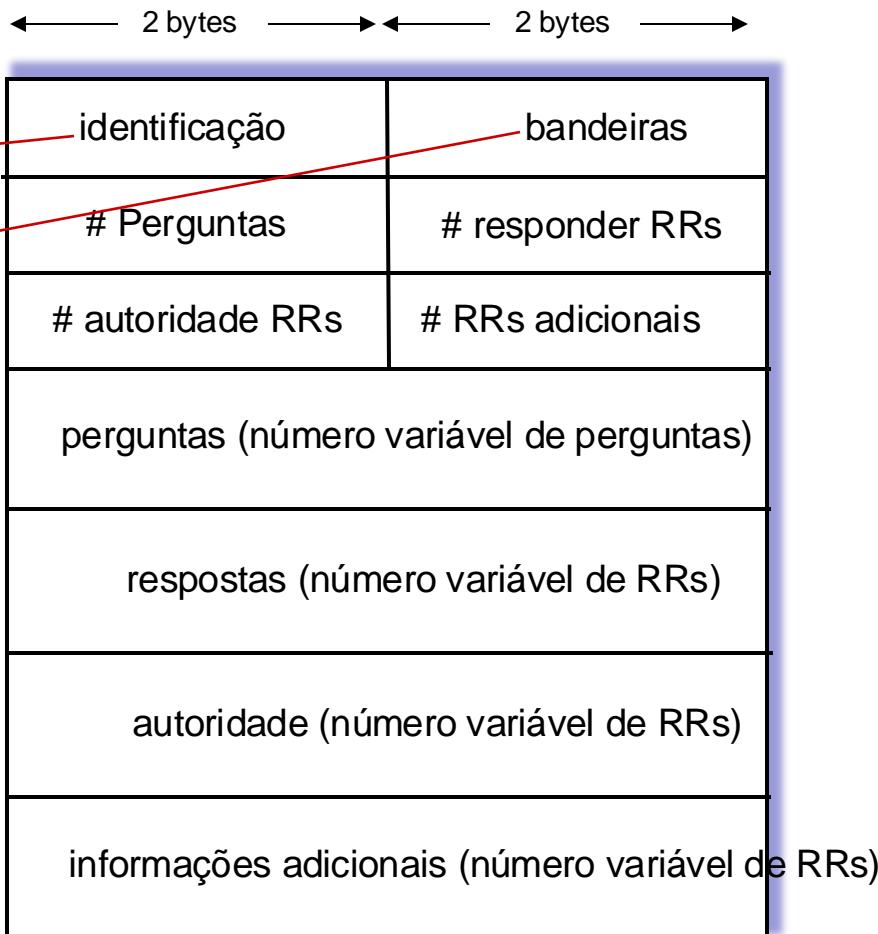
- valor é o nome do servidor de correio eletrônico SMTP associado ao nome

Mensagens do protocolo DNS

Mensagens *de consulta* e *resposta* de DNS, ambas com o mesmo formato:

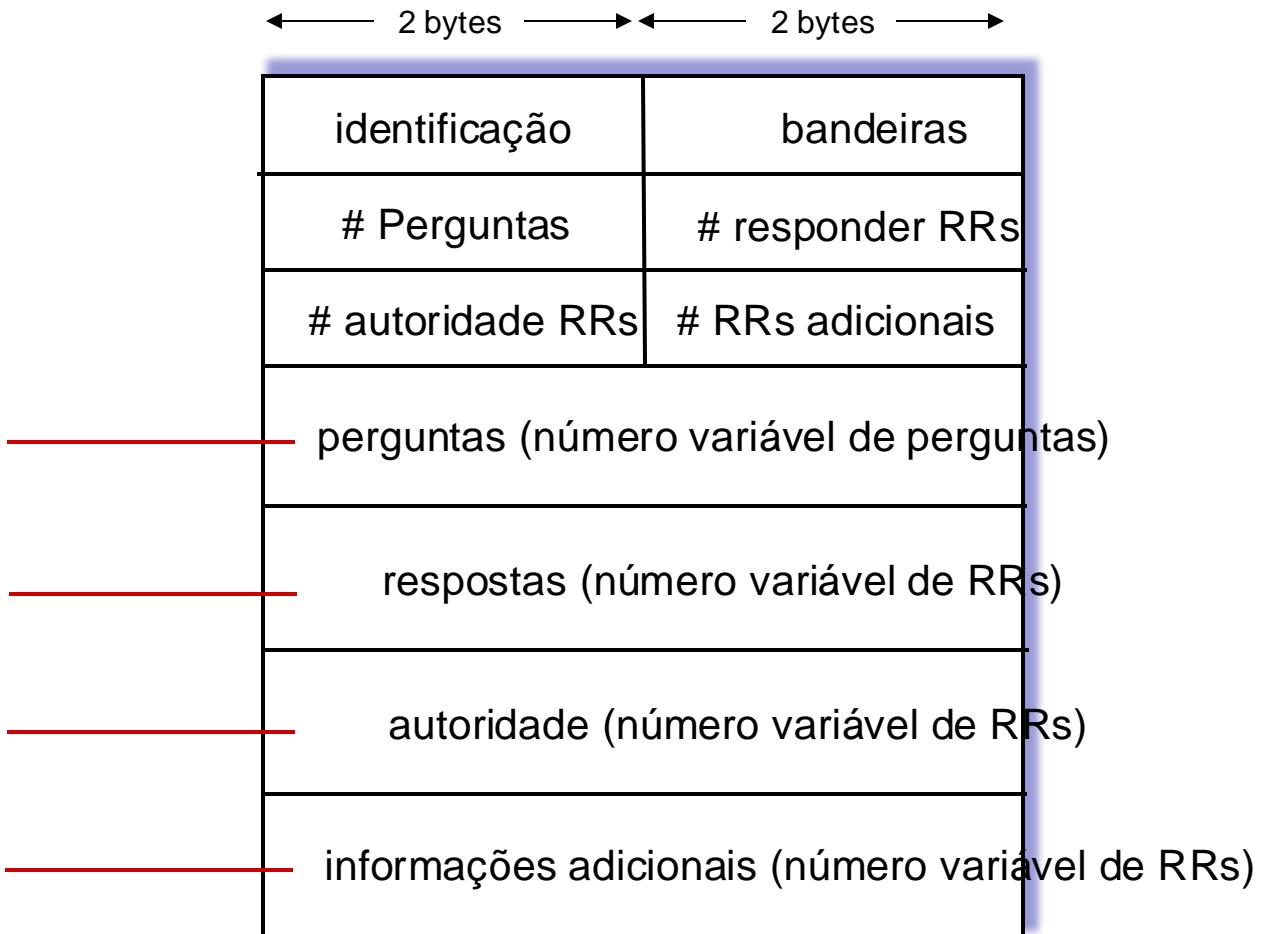
cabeçalho da mensagem:

- identificação: 16 bits # para consulta, a resposta à consulta usa o mesmo #
- bandeiras:
 - consulta ou resposta
 - recursão desejada
 - recursão disponível
 - a resposta é confiável



Mensagens do protocolo DNS

Mensagens *de consulta* e *resposta* de DNS, ambas com o mesmo *formato*:



campos de nome e tipo para
uma consulta

RRs em resposta à consulta

registros para servidores
autoritativos

informações adicionais "úteis" que
podem ser usadas

Colocar suas informações no DNS

exemplo: nova startup "Network Utopia"

- registrar o nome networkuptopia.com no *registrador de DNS* (por exemplo, Network Solutions)
 - Fornecer nomes e endereços IP do servidor de nomes autoritativo (primário e secundário)
 - O registrador insere NS, A RRs no servidor do TLD .com:
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
- criar um servidor autoritativo localmente com o endereço IP 212.212.212.1
 - Registro do tipo A para www.networkuptopia.com
 - digite o registro MX para networkutopia.com

Segurança do DNS

Ataques DDoS

- bombardear os servidores raiz com tráfego
 - sem sucesso até o momento
 - filtragem de tráfego
 - os servidores DNS locais armazenam em cache os IPs dos servidores TLD, permitindo o desvio do servidor raiz
- bombardeiam os servidores de TLDs
 - potencialmente mais perigoso

Ataques de spoofing

- interceptar consultas de DNS, retornando respostas falsas
 - Envenenamento do cache do DNS
 - RFC 4033: Serviços de autenticação DNSSEC

Camada de aplicativos: Visão geral

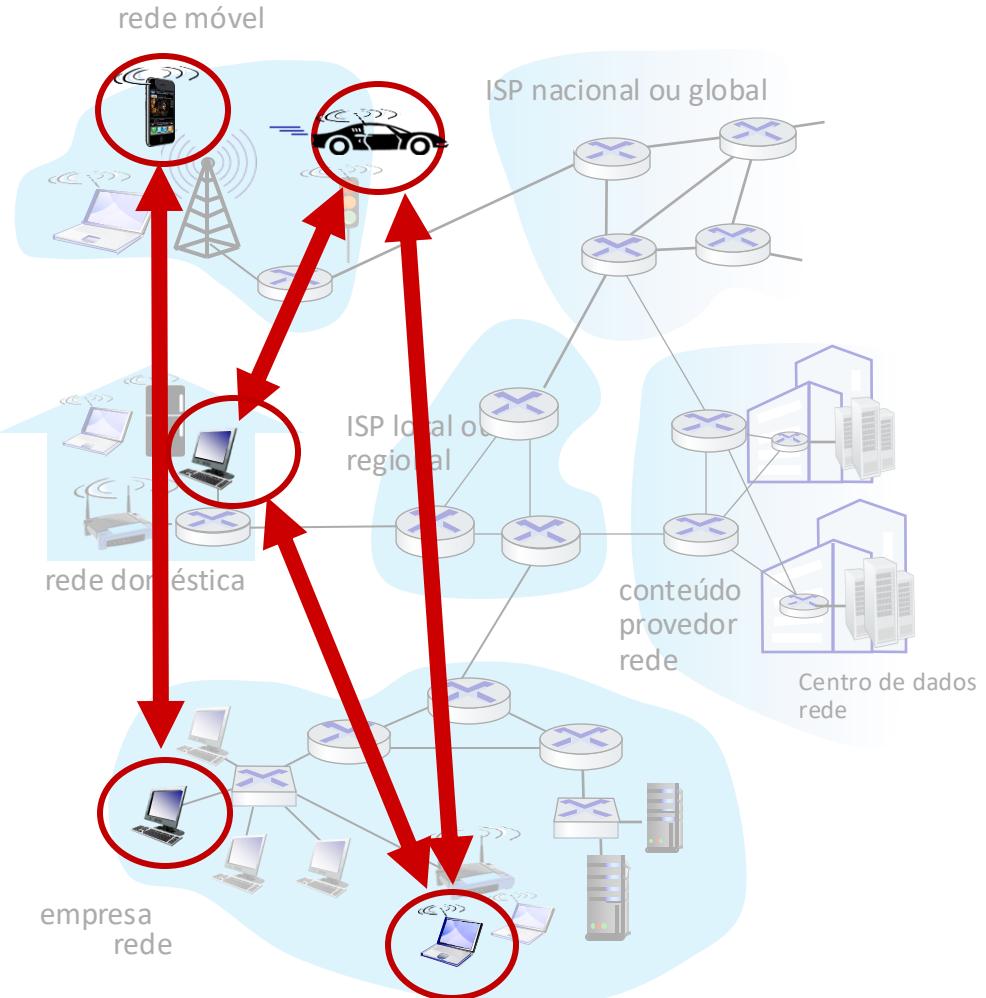
- Princípios de aplicativos de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O sistema de nomes de domínio DNS

- Aplicativos P2P
- redes de distribuição de conteúdo e streaming de vídeo
- programação de soquetes com UDP e TCP



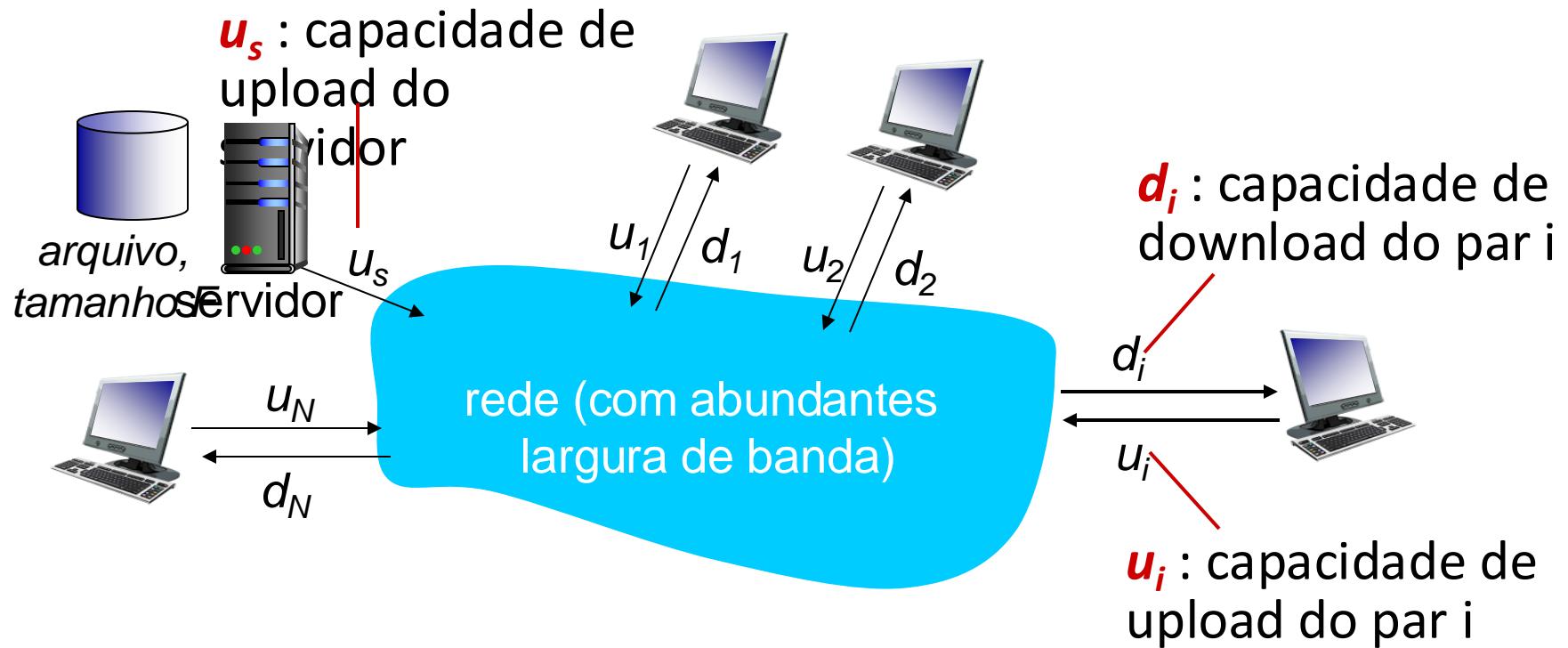
Arquitetura ponto a ponto (P2P)

- *nenhum* servidor sempre ativo
- sistemas finais arbitrários se comunicam diretamente
- os pares solicitam serviços de outros pares e fornecem serviços em troca a outros pares
 - *Autoescalabilidade* - novos pares trazem nova capacidade de serviço e novas demandas de serviço
- os pares estão conectados de forma intermitente e mudam os endereços IP
 - gerenciamento complexo
- Exemplos: Compartilhamento de arquivos P2P (BitTorrent), streaming (KanKan), VoIP (Skype)



Distribuição de arquivos: cliente-servidor vs. P2P

- Q:** quanto tempo leva para distribuir um arquivo (tamanho F) de um servidor para N pares?
- a capacidade de upload/download dos pares é um recurso limitado



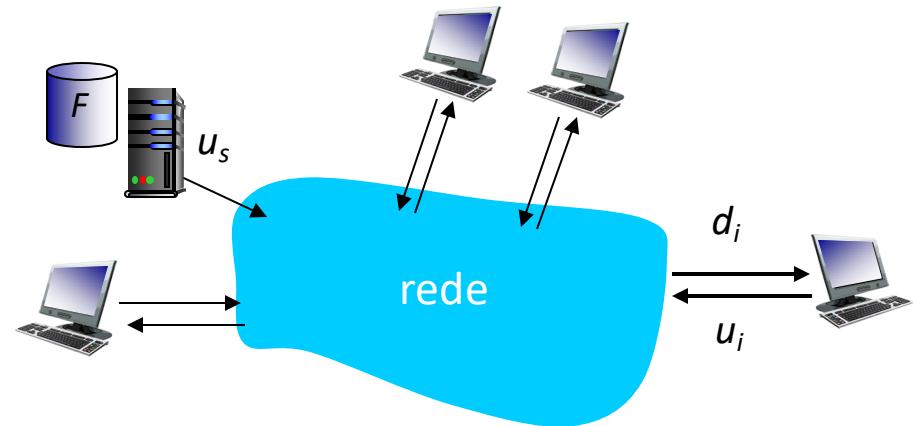
Tempo de distribuição de arquivos: cliente-servidor

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

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

- *cliente*: cada cliente deve fazer o download da cópia do arquivo

- d_{min} = taxa mínima de download do cliente
- tempo mínimo de download 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 arquivos: P2P

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

- tempo para enviar uma cópia: F/u_s

- *cliente:* cada cliente deve fazer o download da cópia do arquivo

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

- *clientes:* como o agregado deve fazer o download dos bits da NF

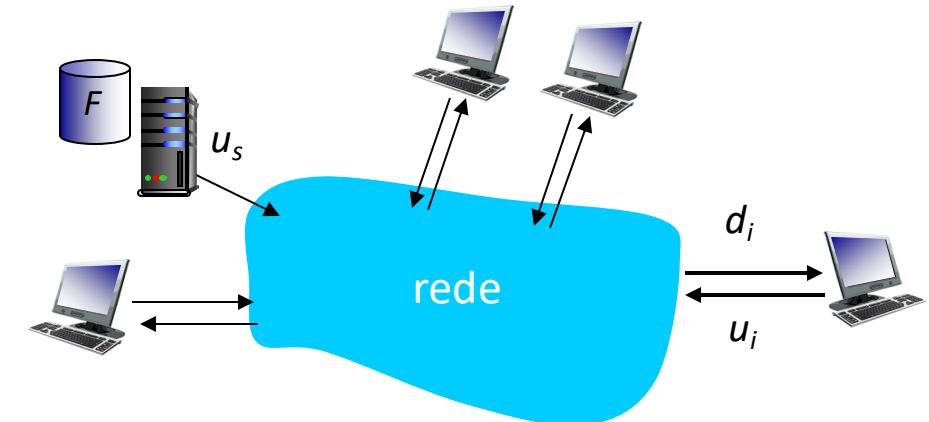
- A taxa máxima de upload (limitando a taxa máxima de download) é $u_s + Su_i$

tempo para distribuir F
para N clientes usando
Abordagem P2P

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

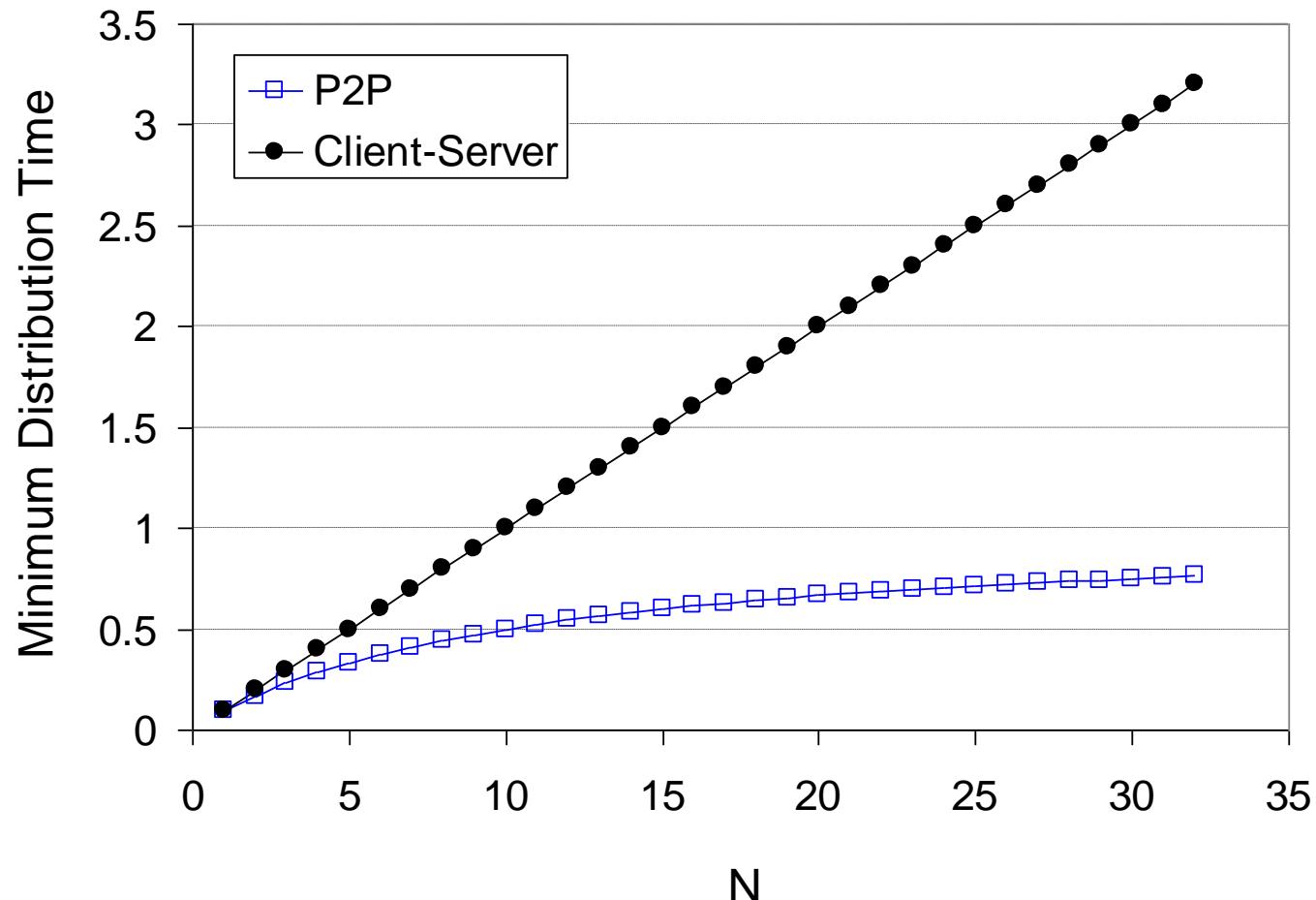
aumenta linearmente em N ...

... mas isso também acontece, pois cada par traz capacidade de serviço



Cliente-servidor vs. P2P: exemplo

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

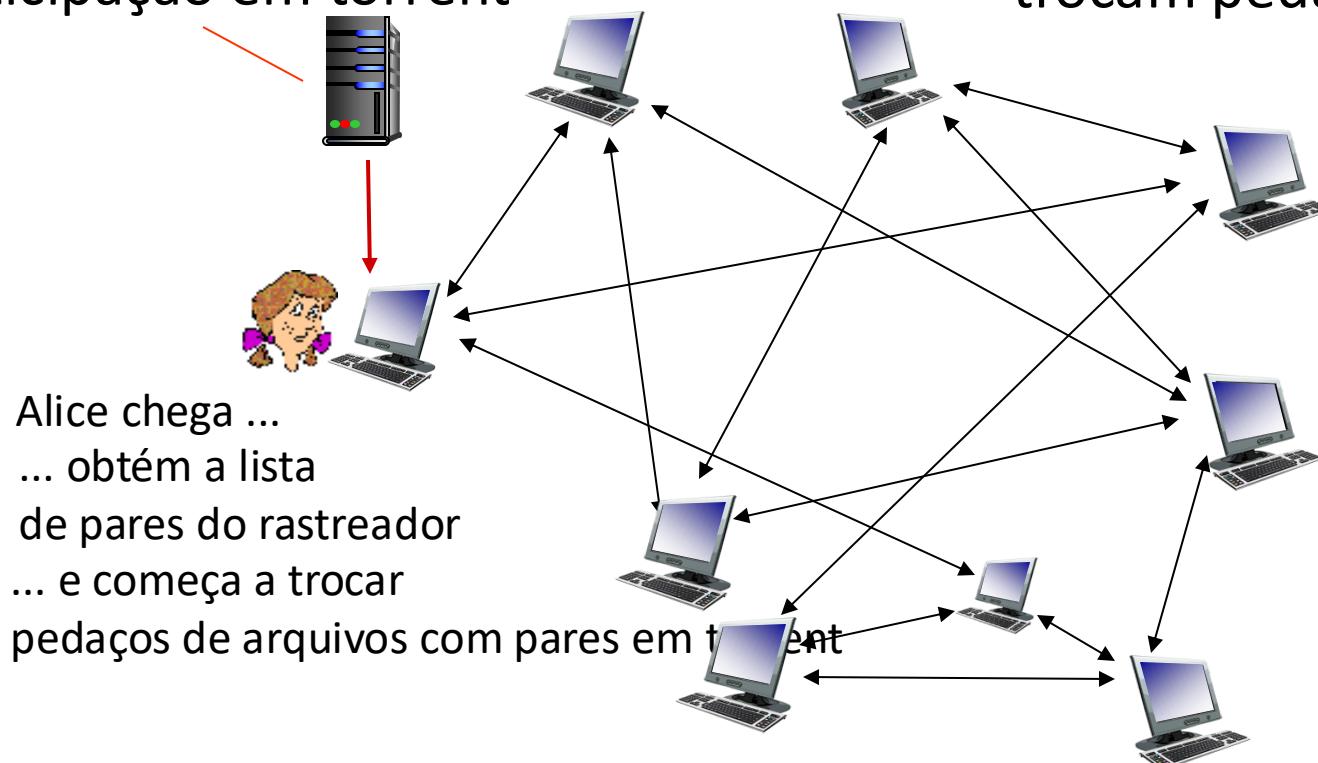


Distribuição de arquivos P2P: BitTorrent

- arquivo dividido em blocos de 256 KB
- pares em torrent enviam/recebem pedaços de arquivos

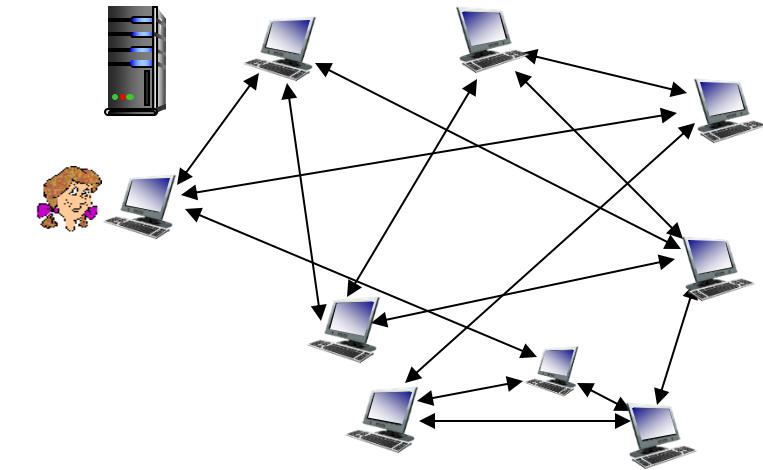
rastreador: rastreia os pares
participação em torrent

torrent: grupo de pares que
trocam pedaços de um arquivo



Distribuição de arquivos P2P: BitTorrent

- torrent de junção de pares:
 - não tem pedaços, mas os acumulará ao longo do tempo a partir de outros pares
 - registra-se no rastreador para obter uma lista de pares, conecta-se a um subconjunto de pares ("vizinhos")
- durante o download, o par faz upload de partes para outros pares
- o par pode mudar os pares com os quais troca pedaços
- *rotatividade*: os pares podem ir e vir
- Quando o par tiver o arquivo inteiro, ele poderá (egoisticamente) sair ou (altruisticamente) permanecer na torrent



BitTorrent: solicitação e envio de blocos de arquivos

Solicitação de blocos:

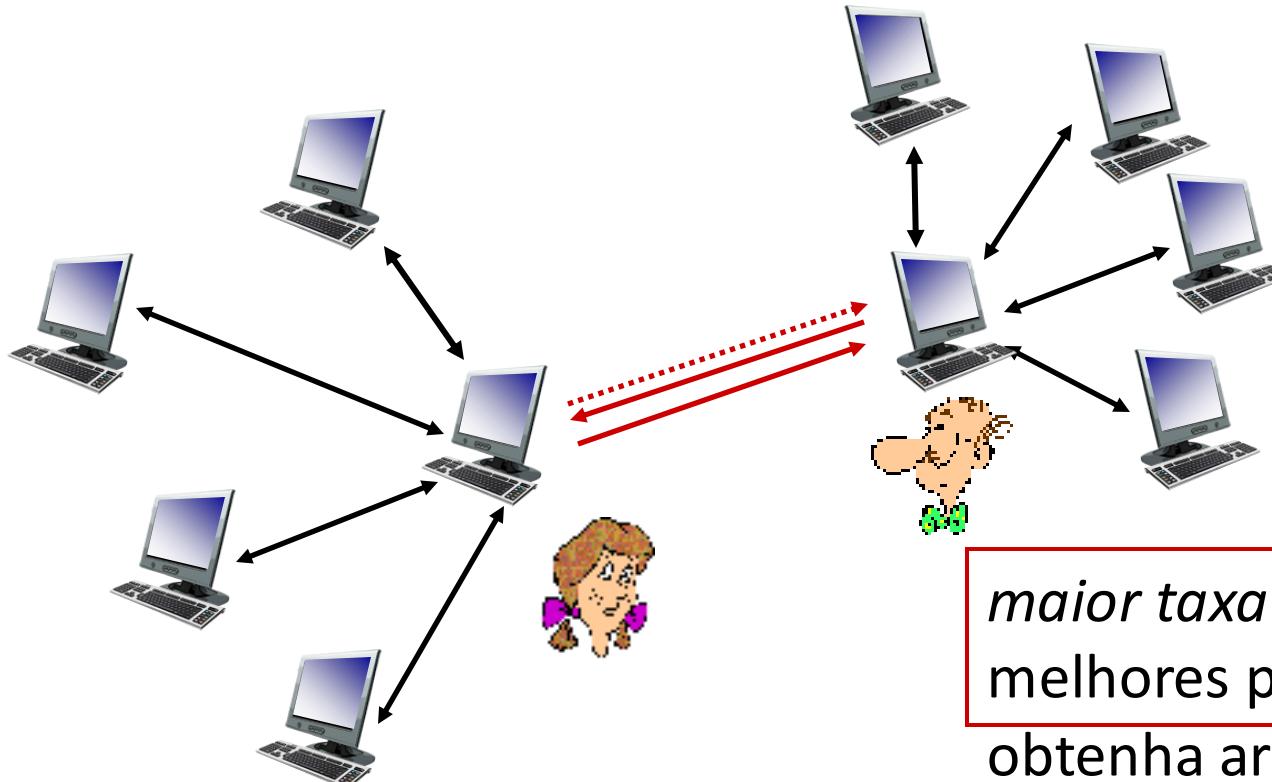
- em um determinado momento, pares diferentes têm subconjuntos diferentes de pedaços de arquivos
- Periodicamente, Alice solicita a cada par a lista de blocos que eles têm
- Alice solicita pedaços ausentes de seus pares, os mais raros primeiro

Envio de blocos: tit-for-tat

- Alice envia pedaços para os quatro pares que estão enviando pedaços para ela *na taxa mais alta*
 - outros pares são bloqueados por Alice (não recebem pedaços dela)
 - reavaliar os 4 primeiros a cada 10 segundos
- a cada 30 segundos: seleciona aleatoriamente outro par e começa a enviar blocos
 - "desvincular de forma otimista" esse par
 - o par recém-escolhido pode entrar no top 4

BitTorrent: tit-for-tat

- (1) Alice "otimisticamente não escolhe" Bob
- (2) Alice se torna um dos quatro principais fornecedores de Bob; Bob retribui
- (3) Bob se torna um dos quatro principais fornecedores de Alice



maior taxa de upload: encontre melhores parceiros comerciais, obtenha arquivos mais rapidamente!

Camada de aplicativos: visão geral

- Princípios de aplicativos de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O sistema de nomes de domínio DNS
- Aplicativos P2P
- **redes de distribuição de conteúdo e streaming de vídeo**
- programação de soquetes com UDP e TCP



Streaming de vídeo e CDNs: contexto

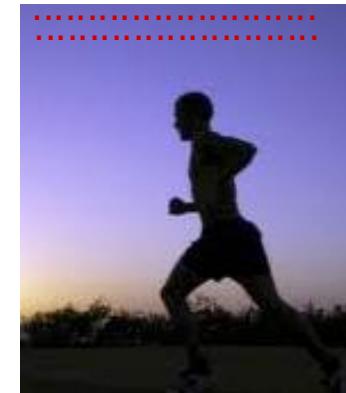
- tráfego de streaming de vídeo: maior consumidor de largura de banda da Internet
 - Netflix, YouTube, Amazon Prime: 80% do tráfego de ISP residencial (2020)
- *desafio*: escala - como atingir ~1 bilhão de usuários?
- *desafio*: heterogeneidade
 - usuários diferentes têm recursos diferentes (por exemplo, com fio versus móvel; com largura de banda rica versus com largura de banda pobre)
- *Solução*: infraestrutura distribuída em nível de aplicativo



Multimídia: vídeo

- vídeo: sequência de imagens exibidas em uma taxa constante
 - Por exemplo, 24 imagens/seg.
- imagem digital: matriz de pixels
 - cada pixel representado por bits
- codificação: usar redundância *dentro* e *entre* imagens para diminuir o número de bits usados para codificar a imagem
 - espacial (dentro da imagem)
 - temporal (de uma imagem para outra)

Exemplo de codificação espacial:
em vez de enviar N valores da mesma cor (todos roxos), envie apenas dois valores: valor da cor (roxo) e *número de valores repetidos* (N)



quadro *i*

Exemplo de codificação temporal: em vez de enviar o quadro completo em $i+1$, envie apenas as diferenças do quadro i

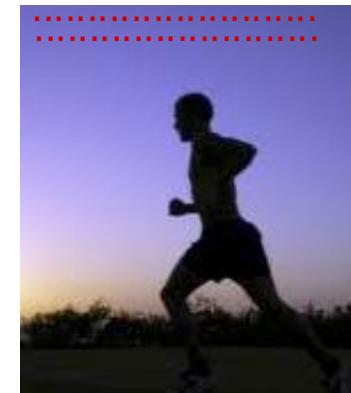


quadro *i+1*

Multimídia: vídeo

- **CBR: (taxa de bits constante):** taxa de codificação de vídeo fixa
- **VBR: (taxa de bits variável):** a taxa de codificação de vídeo muda conforme a quantidade de mudanças na codificação espacial e temporal
- **exemplos:**
 - MPEG 1 (CD-ROM) 1,5 Mbps
 - MPEG2 (DVD) 3-6 Mbps
 - MPEG4 (geralmente usado na Internet, 64 Kbps - 12 Mbps)

Exemplo de codificação espacial:
em vez de enviar N valores da mesma cor (todos roxos), envie apenas dois valores: valor da cor (roxo) e *número de valores repetidos* (N)



quadro *i*

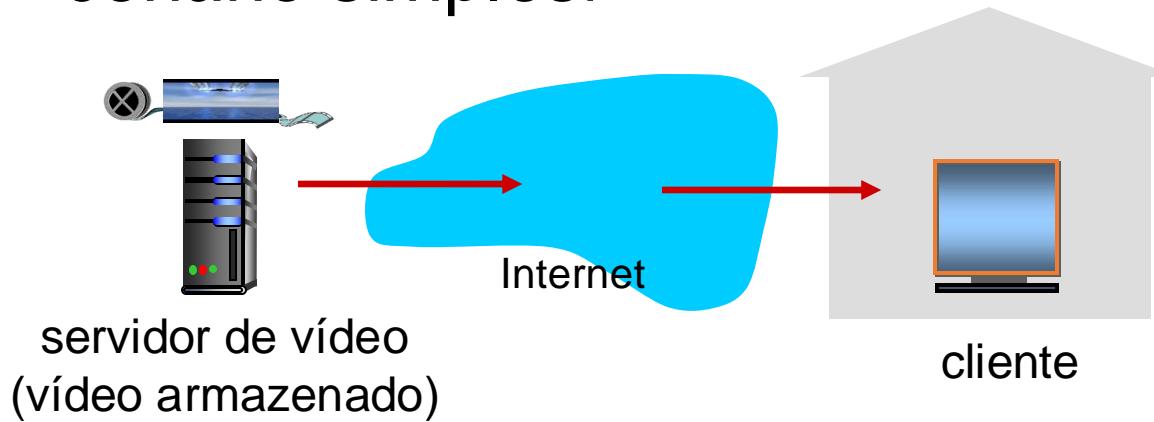
Exemplo de codificação temporal: em vez de enviar o quadro completo em $i+1$, envie apenas as diferenças do quadro i



quadro $i+1$

Transmissão de vídeo armazenado

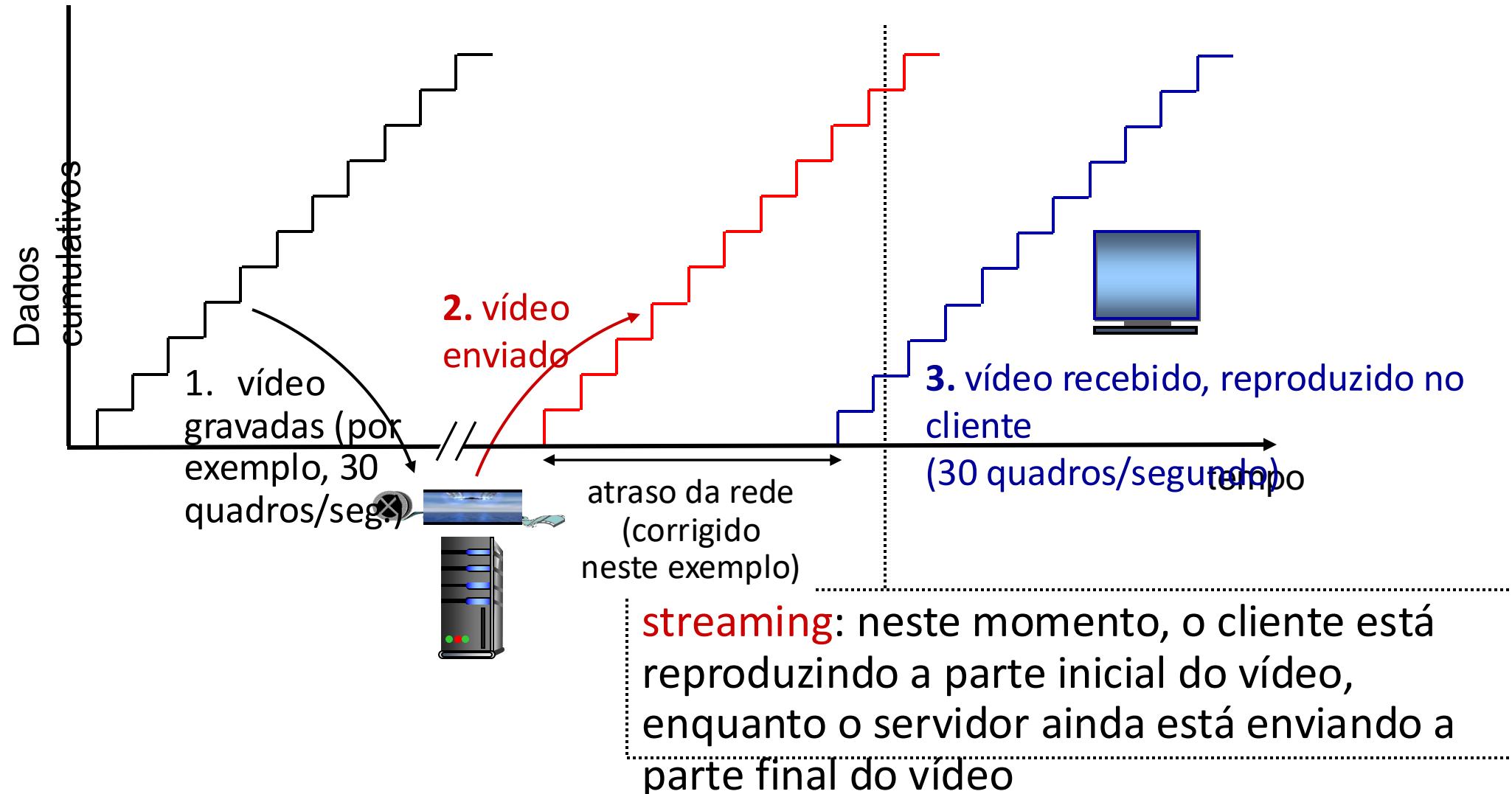
cenário simples:



Principais desafios:

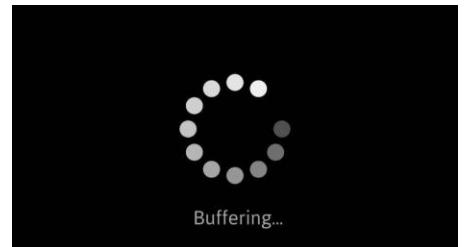
- A largura de banda entre servidor e cliente *varia* com o tempo, com a mudança dos níveis de congestionamento da rede (interna, rede de acesso, núcleo da rede, servidor de vídeo)
- a perda de pacotes, o atraso devido ao congestionamento atrasará a reprodução ou resultará em baixa qualidade de vídeo

Transmissão de vídeo armazenado

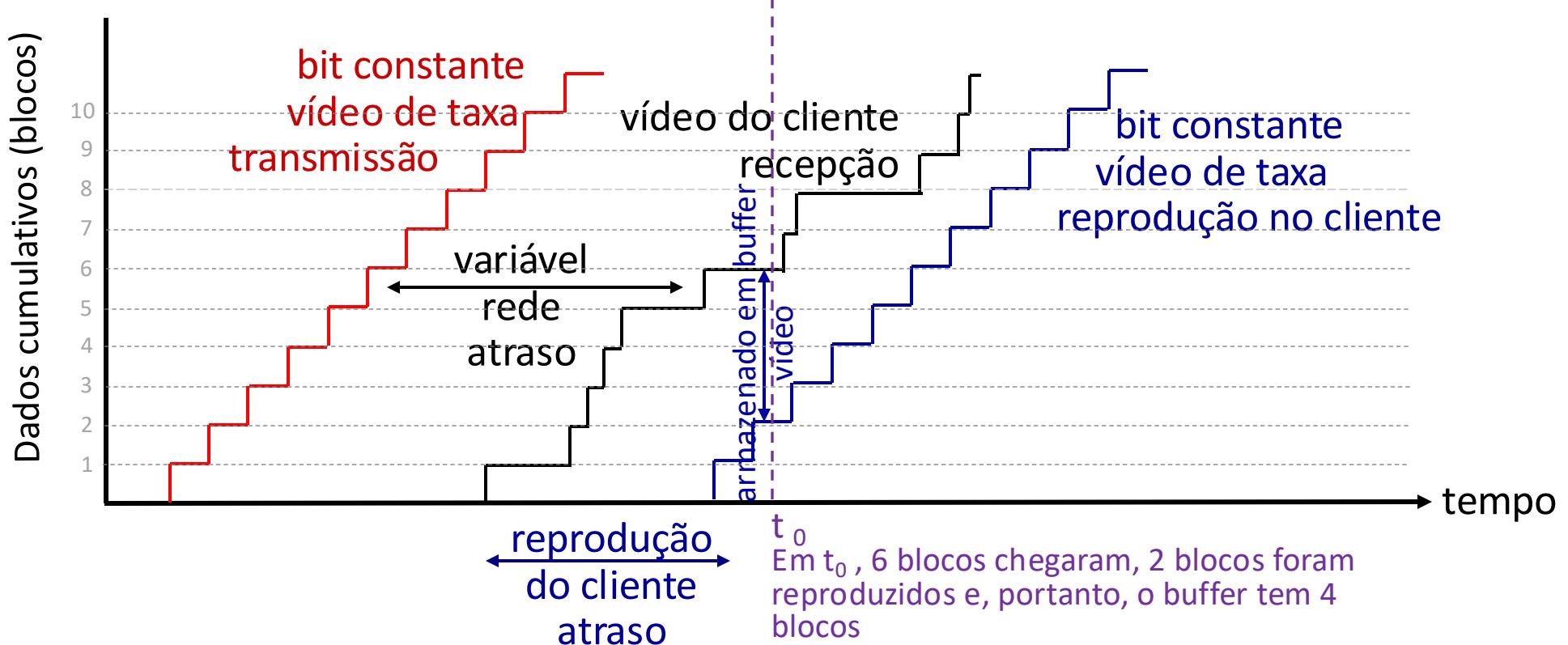


Streaming de vídeo armazenado: desafios

- Restrição de reprodução contínua: durante a reprodução do vídeo do cliente, o tempo de reprodução deve corresponder ao tempo original
 - ... mas os atrasos na rede são variáveis (jitter), portanto, será necessário um buffer no lado do cliente para corresponder à restrição de reprodução contínua
- outros desafios:
 - Interatividade do cliente: pausar, avançar, retroceder, pular pelo vídeo
 - os pacotes de vídeo podem ser perdidos, retransmitidos



Streaming de vídeo armazenado: buffer de reprodução



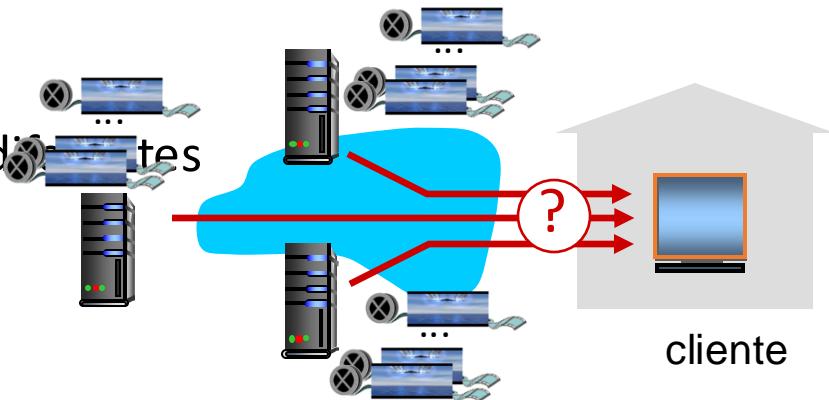
- *buffering no lado do cliente e atraso de reprodução:* compensar o atraso adicionado pela rede, jitter de atraso

Transmissão de multimídia: DASH

Streaming dinâmico e adaptável por HTTP

servidor:

- divide o arquivo de vídeo em vários blocos
- cada bloco codificado em várias taxas diferentes
- codificações de taxas diferentes armazenadas em arquivos diferentes
- arquivos replicados em vários nós de CDN
- *arquivo de manifesto*: fornece URLs para diferentes blocos

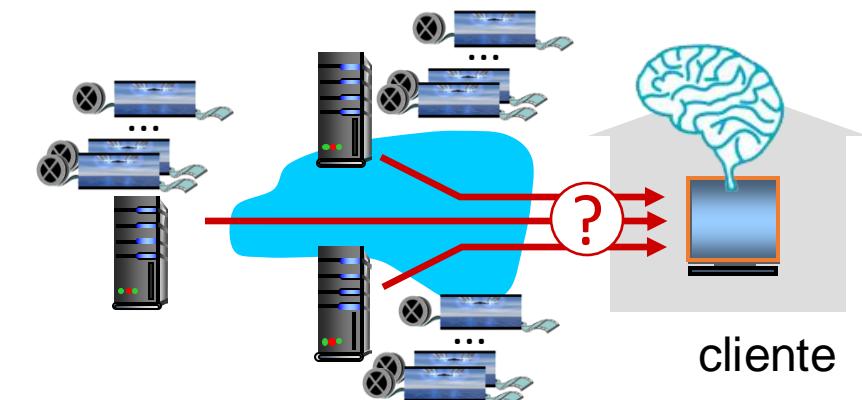


cliente:

- estima periodicamente a largura de banda de servidor para cliente
- Consultar o manifesto, solicitar uma parte de cada vez
 - escolhe a taxa de codificação máxima sustentável dada a largura de banda atual
 - pode escolher diferentes taxas de codificação em diferentes momentos (dependendo da largura de banda disponível no momento) e de diferentes servidores

Transmissão de multimídia: DASH

- "*inteligência*" no cliente: o cliente determina
 - *quando* solicitar um bloco (para que não ocorra falta de buffer ou estouro)
 - *qual taxa de codificação* solicitar (maior qualidade quando houver mais largura de banda disponível)
 - *onde* solicitar o bloco (pode solicitar do servidor de URL que esteja "próximo" do cliente ou que tenha alta largura de banda disponível)



Streaming de vídeo = codificação + DASH + buffer de reprodução

Redes de distribuição de conteúdo (CDNs)

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

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

.... de forma bastante simples: essa solução *não é escalonável*

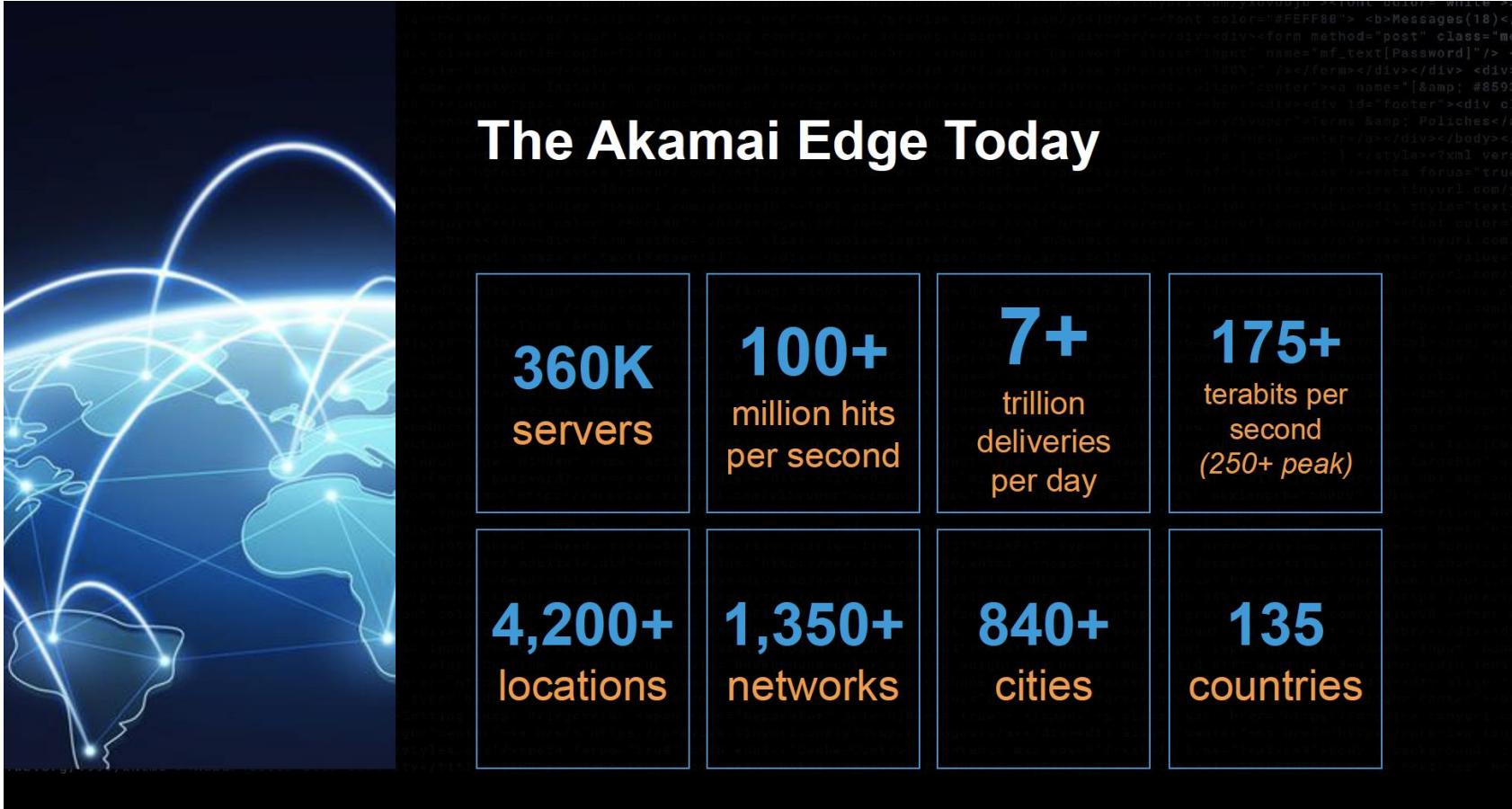
Redes de distribuição de conteúdo (CDNs)

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 locais distribuídos geograficamente (**CDN**)
 - *entrar em profundidade:* empurre os servidores CDN para dentro de muitas redes de acesso
 - próximo aos usuários
 - Akamai: 240.000 servidores implantados em mais de 120 países (2015)
 - *Trazer para casa:* menor número (10) de grupos maiores em POPs próximos a redes de acesso
 - usado pela Limelight



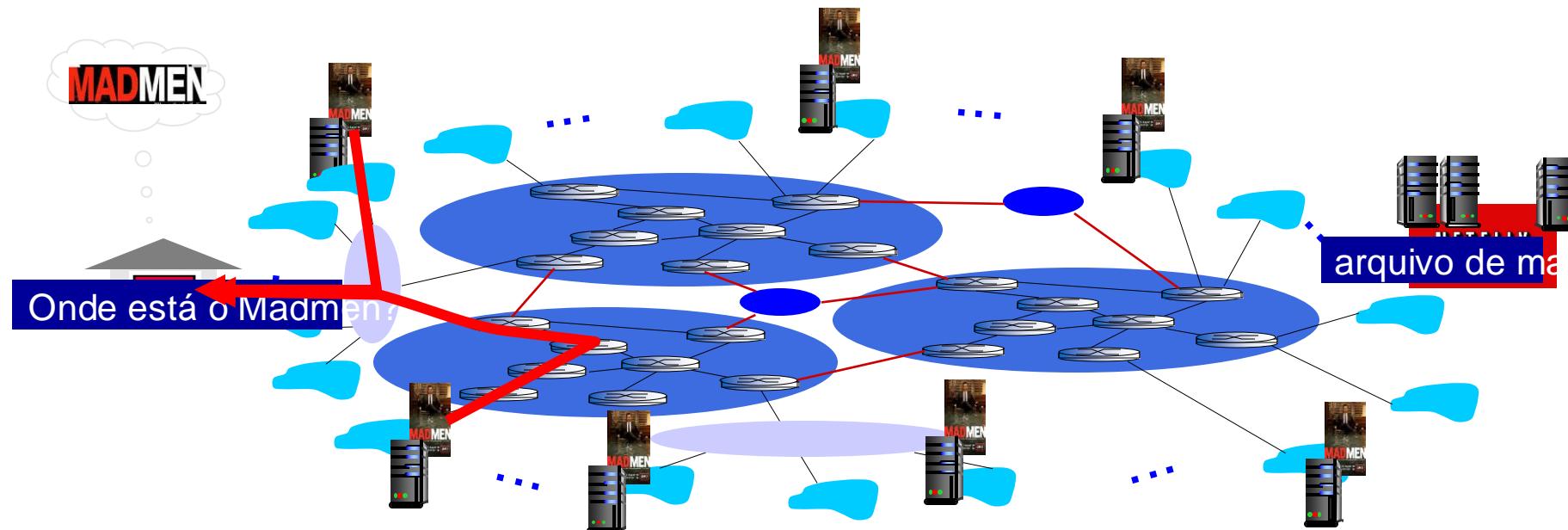
Akamai hoje:



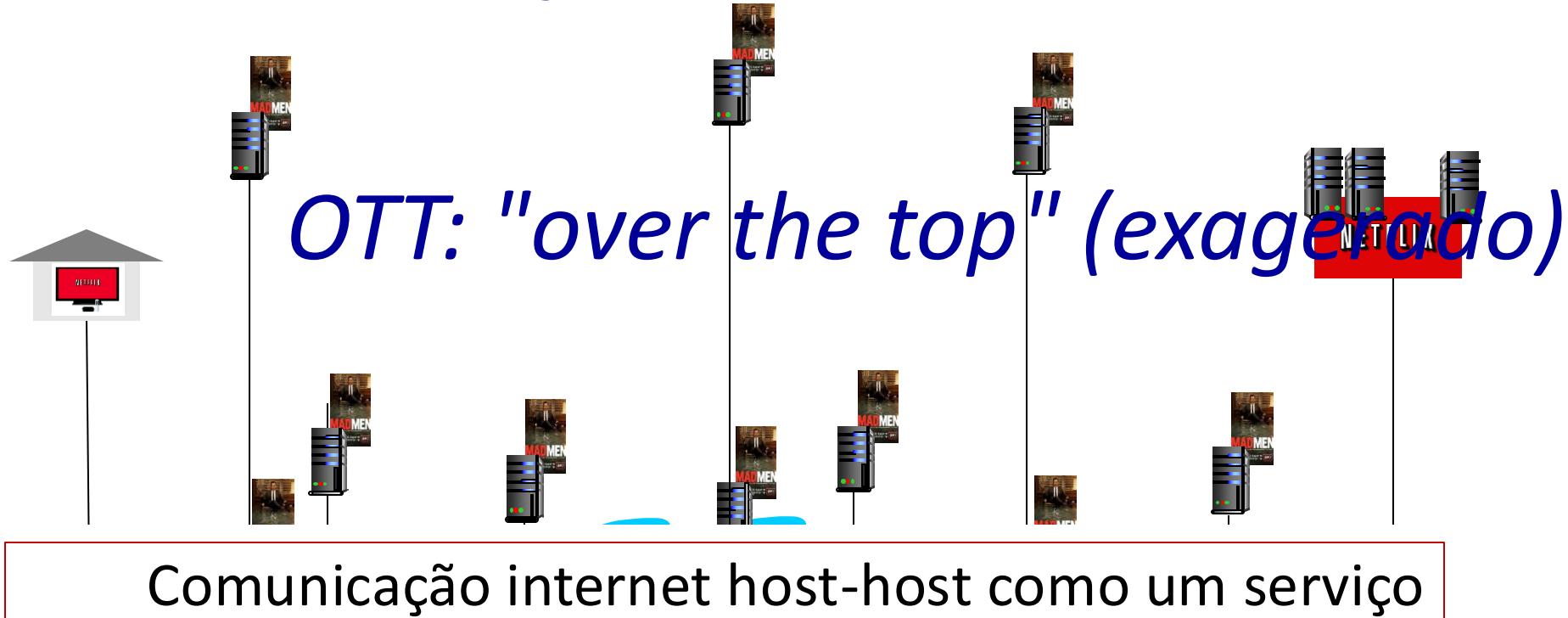
Fonte: <https://networkingchannel.eu/living-on-the-edge-for-a-quarter-century-an-akamai-retrospective-downloads/>

Como funciona a Netflix?

- Netflix: armazena cópias de conteúdo (por exemplo, MADMEN) em seus nós OpenConnect CDN (em todo o mundo)
- O assinante solicita conteúdo, o provedor de serviços retorna o manifesto
 - usando o manifesto, o cliente recupera o conteúdo na taxa mais alta
 - pode escolher uma taxa diferente ou copiar se o caminho da rede estiver congestionado



Redes de distribuição de conteúdo (CDNs)



Desafios OTT: lidando com uma Internet congestionada a partir da "borda"

- Qual conteúdo colocar em qual nó de CDN?
- de qual nó da CDN para recuperar o conteúdo? A que taxa?

Camada de aplicativos: Visão geral

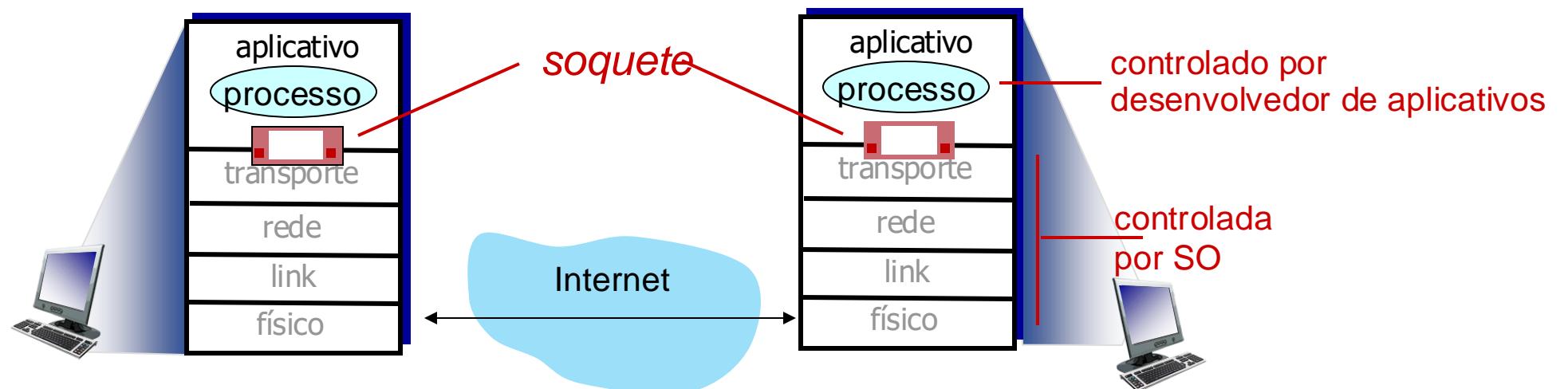
- Princípios de aplicativos de rede
- Web e HTTP
- E-mail, SMTP, IMAP
- O sistema de nomes de domínio DNS
- Aplicativos P2P
- redes de distribuição de conteúdo e streaming de vídeo
- **programação de soquetes com UDP e TCP**



Programação de soquetes

Objetivo: aprender a criar aplicativos cliente/servidor que se comunicam usando soquetes

soquete: porta entre o processo do aplicativo e o protocolo de transporte final



Programação de soquetes

Dois tipos de soquete para dois serviços de transporte:

- *UDP*: datagrama não confiável
- *TCP*: confiável, orientado para o fluxo de bytes

Exemplo de aplicativo:

1. o cliente lê uma linha de caracteres (dados) em seu teclado e envia os dados para o servidor
2. O servidor recebe os dados e converte os caracteres em letras maiúsculas
3. o servidor envia dados modificados ao cliente
4. o cliente recebe dados modificados e exibe a linha em sua tela

Programação de soquetes com UDP

UDP: não há "conexão" entre o cliente e o servidor:

- sem handshaking antes de enviar dados
- o remetente anexa explicitamente o endereço IP de destino e a porta nº a cada pacote
- o receptor extrai o endereço IP do remetente e o número da porta do pacote recebido

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

Ponto de vista do aplicativo:

- O UDP oferece transferência *não confiável* de grupos de bytes ("datagramas") entre processos de cliente e servidor

Interação de soquete cliente/servidor: UDP



servidor (em execução no IP do servidor)

```
criar soquete, porta= x:  
serverSocket =  
socket(AF_INET,SOCK_DGRAM)
```

ler datagrama de
servidorSocket

escrever resposta a
servidorSocket
especificando
endereço do cliente,
número da porta



cliente

```
criar soquete:  
clientSocket =  
socket(AF_INET,SOCK_DGRAM)
```

Criar datagrama com endereço IP do servidor
e port=x; enviar datagrama via
clientSocket

ler datagrama de
clienteSocket
próximo
clienteSocket

Exemplo de aplicativo: Cliente UDP

Cliente Python UDPClient

```
incluir a biblioteca de soquetes do Python → from socket import *
serverName = 'hostname' (nome do host)
serverPort = 12000
criar soquete UDP → clientSocket = socket(AF_INET,
                                             SOCK_DGRAM)
obter entrada de teclado do usuário → message = input('Input lowercase sentence:')
anexar o nome do servidor e a porta à mensagem; enviar para o soquete → clientSocket.sendto(message.encode(),
                                            (serverName, serverPort))
ler dados de resposta (bytes) do soquete → modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)
imprime a string recebida e fecha o soquete → print(modifiedMessage.decode())
clientSocket.close()
```

Exemplo de aplicativo: Servidor UDP

Python UDPServer

```
from socket import *
serverPort = 12000
criar soquete UDP → serverSocket = socket(AF_INET, SOCK_DGRAM)
vincular o soquete à porta local número → serverSocket.bind("", serverPort)
12000
print('O servidor está pronto para receber')
loop para → enquanto True:
sempre → message, clientAddress = serverSocket.recvfrom(2048)
Ler do soquete UDP para a mensagem, → modifiedMessage = message.decode().upper()
obtendo o endereço do cliente (IP e porta do → serverSocket.sendto(modifiedMessage.encode(),
cliente) → clientAddress)
enviar string em maiúsculas de volta para → esse cliente
```

Programação de soquetes com TCP

O cliente deve entrar em contato com o servidor

- o processo do servidor deve estar em execução primeiro
- O servidor deve ter criado um soquete (porta) que receba o contato do cliente

O cliente entra em contato com o servidor por:

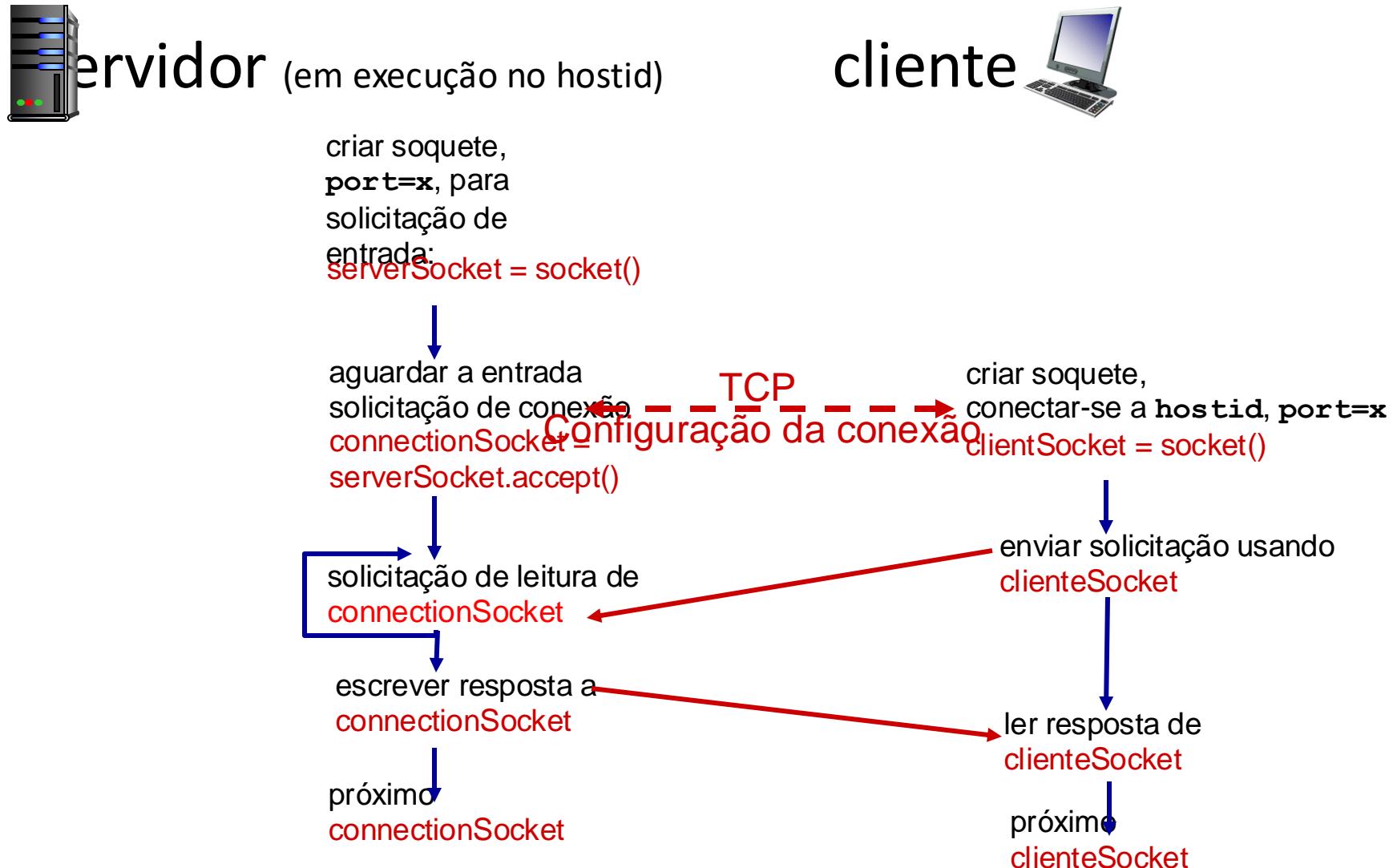
- Criação de soquete TCP, especificando o endereço IP e o número da porta do processo do servidor
- *quando o cliente cria o soquete:* o TCP do cliente estabelece conexão com o TCP do servidor

- Quando contatado pelo cliente, *o TCP do servidor cria um novo soquete* para que o processo do servidor se comunique com aquele cliente específico
 - permite que o servidor se comunique com vários clientes
 - número da porta de origem do cliente e endereço IP usados para distinguir os clientes (mais informações no Capítulo 3)

Ponto de vista do aplicativo

○ TCP oferece um serviço comutável e em ordem
transferência de fluxo de bytes ("pipe")
entre os processos do cliente e do servidor

Interação de soquete cliente/servidor: TCP



Exemplo de aplicativo: Cliente TCP

TCPCliente Python

criar soquete TCP para o
servidor, porta remota 12000



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

Não há necessidade de anexar o nome
do servidor, a porta



Exemplo de aplicativo: Servidor TCP

Servidor TCPS Python

criar soquete TCP de boas-vindas → from socket import *

o servidor começa a escutar as solicitações TCP de entrada → serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(("",serverPort))
serverSocket.listen(1)

loop para sempre → print('O servidor está pronto para receber')

o servidor espera em accept() por solicitações de entrada, novo soquete criado no retorno → enquanto True:
connectionSocket, addr = serverSocket.accept()

ler bytes do soquete (mas não o endereço, como no UDP) → sentença = connectionSocket.recv(1024).decode()
capitalizedSentence = sentence.upper()
connectionSocket.send(capitalizedSentence.encode())

fechar a conexão com esse cliente (mas não o soquete de boas-vindas) → connectionSocket.close()

Capítulo 2: Resumo

nosso estudo sobre a camada de aplicativos de rede está concluído!

- arquiteturas de aplicativos
 - cliente-servidor
 - P2P
- requisitos de serviço do aplicativo:
 - confiabilidade, largura de banda, atraso
- Modelo de serviço de transporte da Internet
 - orientado à conexão, confiável: TCP
 - não confiáveis, datagramas: UDP
- protocolos específicos:
 - HTTP
 - SMTP, IMAP
 - DNS
 - P2P: BitTorrent
- streaming de vídeo, CDNs
- programação de soquetes:
Soquetes TCP, UDP

Capítulo 2: Resumo

E o mais importante: aprendi sobre *protocolos!*

- troca típica de mensagens de solicitação/resposta:
 - o cliente solicita informações ou serviços
 - O servidor responde com dados, código de status
- formatos de mensagem:
 - *cabeçalhos*: campos que fornecem informações sobre os dados
 - *dados*: informações (carga útil) que estão sendo comunicadas

temas importantes:

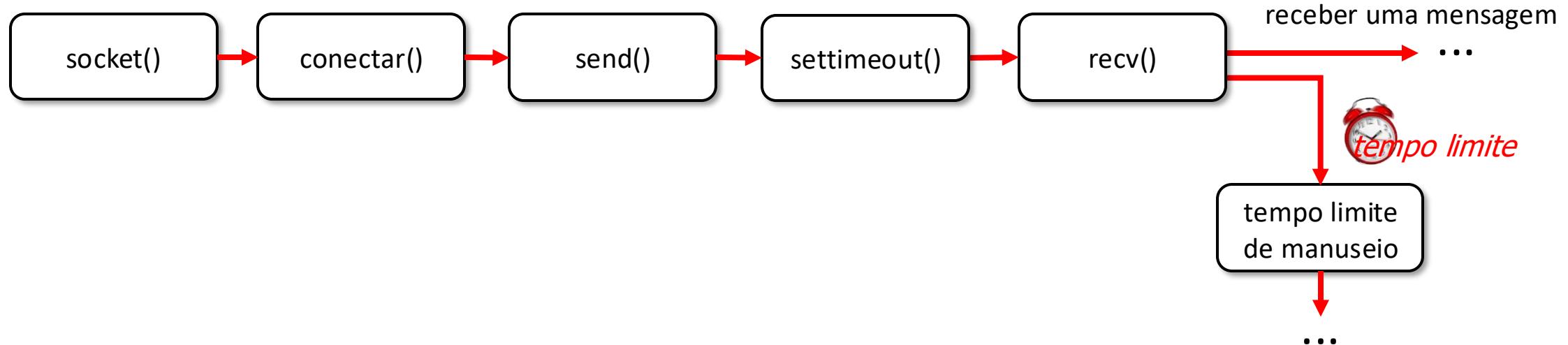
- centralizado vs. descentralizado
- sem estado vs. com estado
- escalabilidade
- transferência de mensagens confiável vs. não confiável
- "complexidade na borda da rede"

Slides adicionais do Capítulo 2

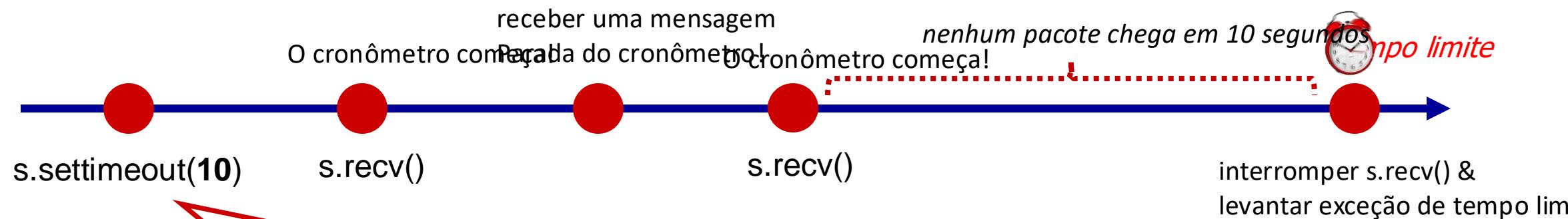
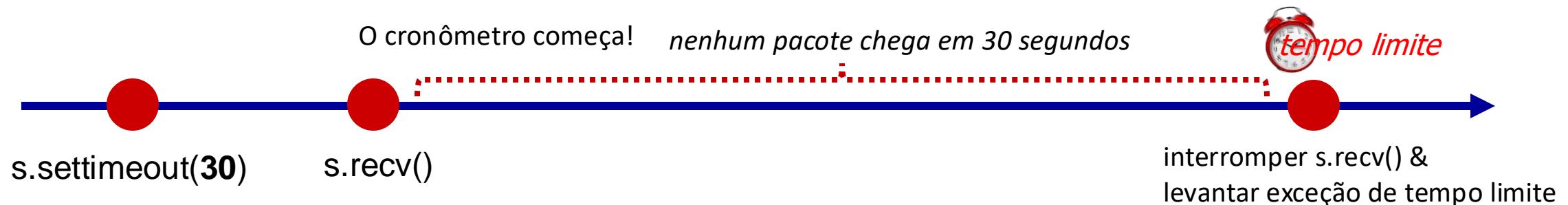
Nota do JFK: os slides de tempo limite são importantes, na minha opinião, se alguém estiver fazendo um trabalho de programação (especialmente um trabalho de programação RDT no Capítulo 3), pois os alunos precisarão usar temporizadores em seu código, e o TRY/EXCEPT é realmente a maneira mais fácil de fazer isso. Apresento isso aqui no Capítulo 2 com o exercício de programação de soquete, pois ensina algo (como lidar com exceções/timeouts) e permite que os alunos aprendam/pratiquem isso antes de fazer o exercício de programação RDT, que é mais difícil

Programação de soquetes: aguardando vários eventos

- Às vezes, um programa precisa **aguardar** a ocorrência de **um dos vários eventos**, por exemplo:
 - aguardar (i) uma resposta de outra extremidade do soquete ou (ii) o tempo limite: timer
 - aguardar respostas de vários soquetes abertos diferentes: `select()`, multithreading
- Os tempos limite são amplamente usados em redes
- usando tempos limite com o soquete Python:



Como funciona o socket.settimeout() do Python?



Defina um tempo limite para todas as futuras operações de soquete desse soquete específico!

Bloco try-except do Python

Executar um bloco de código e tratar as "exceções" que podem ocorrer durante a execução desse bloco de código

tentar:

<fazer algo>

exceto <exceção>:

<tratar a exceção>

A execução desse **bloco de código try** pode causar a captura de exceções. Se uma exceção for levantada, a execução salta de salta diretamente para o **bloco de código except**

Esse **bloco de código except** só é executado se ocorrer uma **<exceção>** no **bloco de código try** (observação: o bloco except é *necessário* com um bloco try)

Programação de soquetes: tempos limite de soquetes

Exemplo de brinquedo:



- Um menino pastor cuida das ovelhas de seu mestre.
- Se ele vir um lobo, poderá enviar uma mensagem aos aldeões pedindo ajuda usando um soquete TCP.
- O garoto achou divertido se conectar ao servidor sem enviar nenhuma mensagem. Mas os aldeões não pensam assim.
- E eles decidiram que, se o garoto se conectar ao servidor e não enviar a localização do lobo **dentro de 10 segundos por três vezes**, eles **deixarão de ouvi-lo para todo o sempre**.

definir um tempo limite de 10 segundos para todas as operações futuras do soquete

O cronômetro começa quando recv() é chamada e levantará uma exceção de tempo limite se não houver nenhuma mensagem dentro de 10 segundos.

capturar a exceção de tempo limite do soquete

Python TCPServer (Aldeões)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(("",serverPort))
serverSocket.listen(1)
contador = 0
enquanto o contador < 3:
    connectionSocket, addr = serverSocket.accept()
    connectionSocket.settimeout(10)
    tentar:
        wolf_location = connectionSocket.recv(1024).decode()
        send_hunter(wolf_location) # uma função de aldeão
        connectionSocket.send('hunter sent')
    exceto o tempo limite:
        contador += 1
        connectionSocket.close()
```

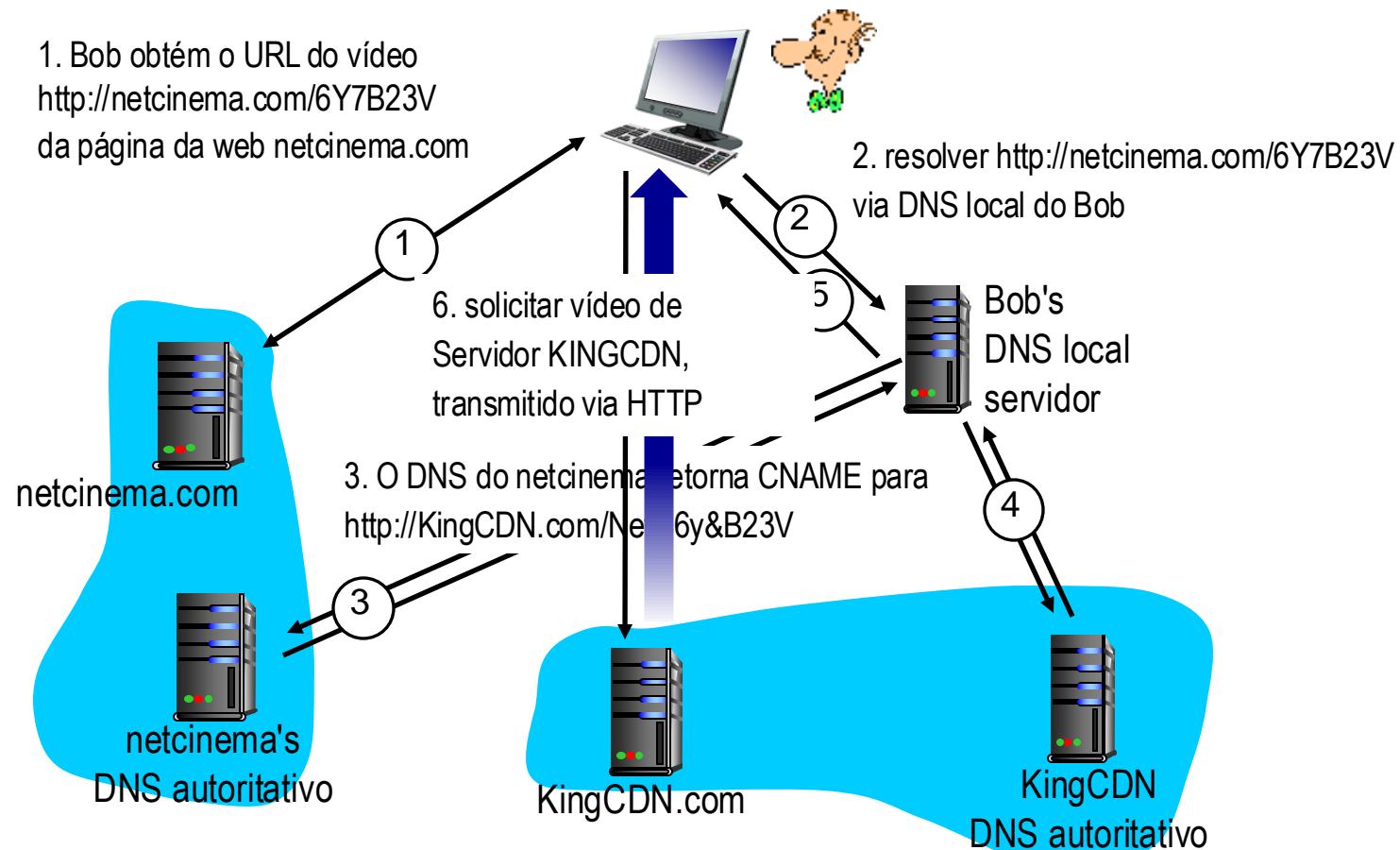
Exemplo de interação SMTP

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, prazer em conhecê-lo
C: E-MAIL DE: <alice@crepes.fr>
S: 250 alice@crepes.fr... Remetente ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Destinatário ok
C: DADOS
S: 354 Digite o e-mail, termine com "." em uma linha separada
C: Você gosta de ketchup?
C: E quanto aos picles?
C: .
S: 250 Mensagem aceita para entrega
C: SAIR
S: 221 hamburger.edu fechando conexão
```

Acesso ao conteúdo da CDN: uma análise mais detalhada

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

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



Estudo de caso: Netflix

