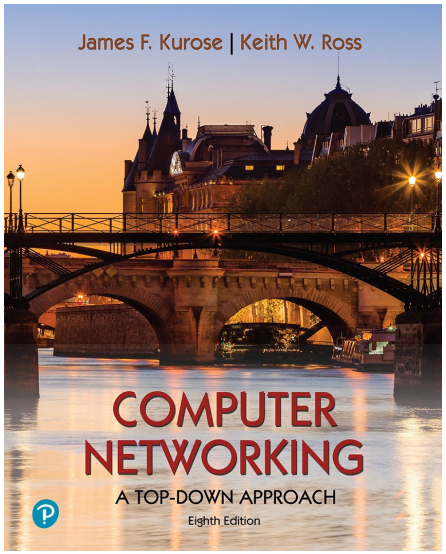
UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Camada de Aplicações - Parte 01

GEX105 - REDES DE COMPUTADORES

Livro-Texto:



REDES DE COMPUTADORES E A INTERNET
6ª Edição
James F. Kurose e Keith W. Ross
Copyright: 2014
656 páginas - ISBN: 9788581436777

Baseado nos slides de Kurose e Ross

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

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

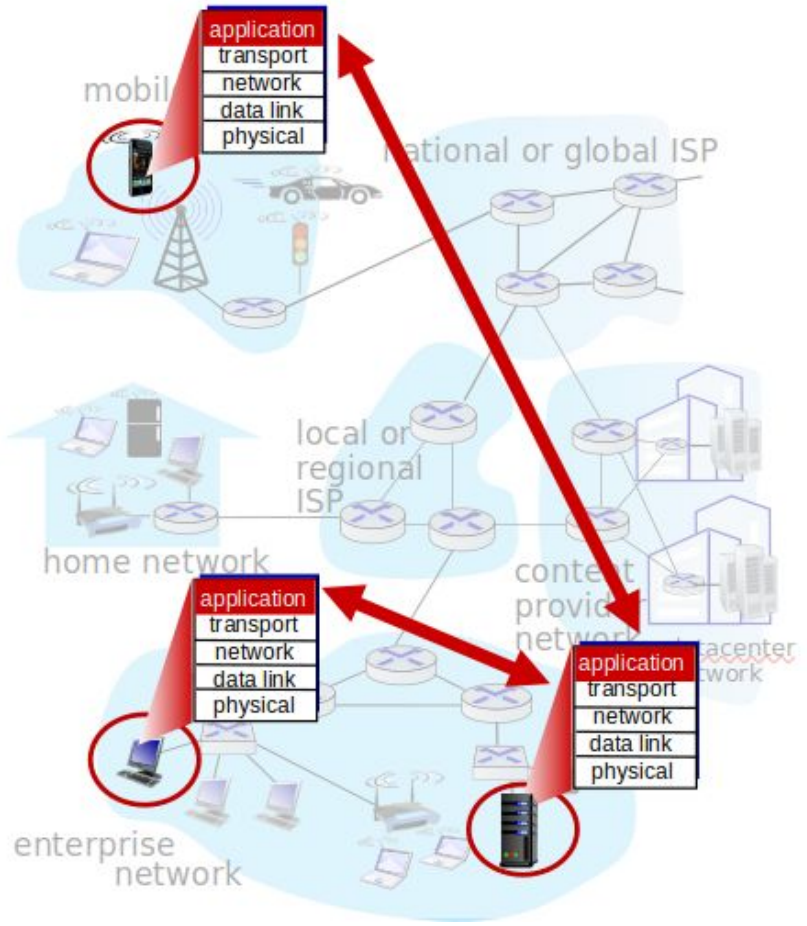
- Redes sociais
- Web
- Mensagens de texto
- e-mail
- Jogos de rede multiusuário
- Transmissão de vídeo armazenado (YouTube, Netflix)
- Compartilhamento de arquivos P2P
- voice over IP (e.g., Skype)
- real-time video conferencing (e.g., Zoom)
- Internet search
- remote login
- …

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

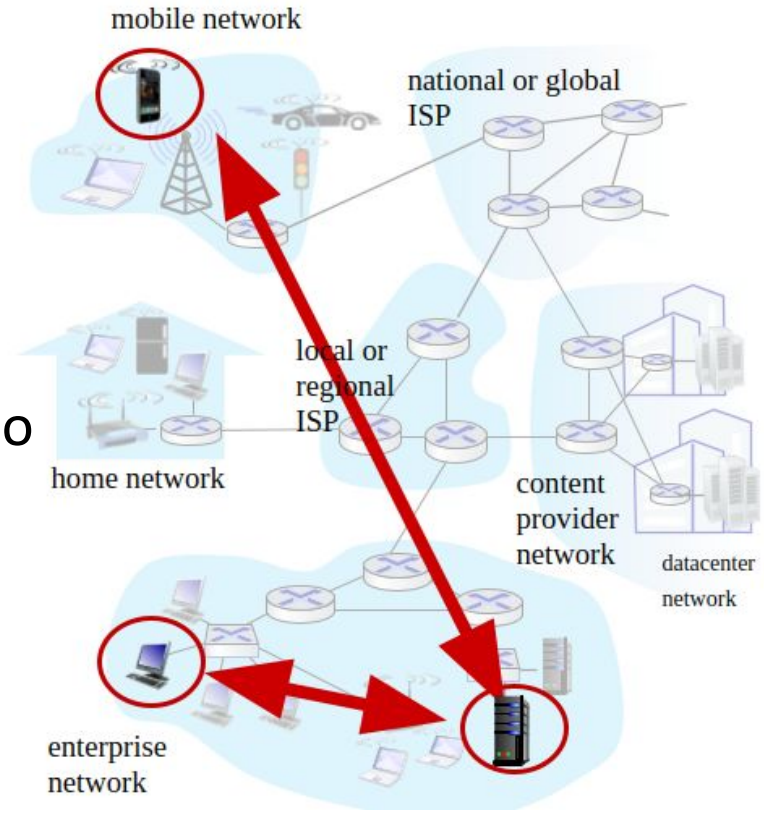
Paradigma Cliente Servidor

Servidor:

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

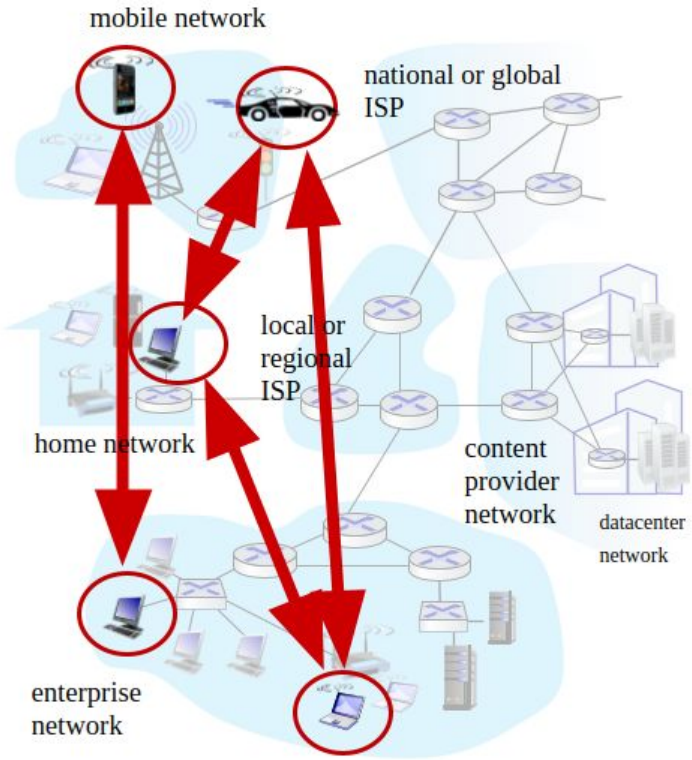
Clientes:

- 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

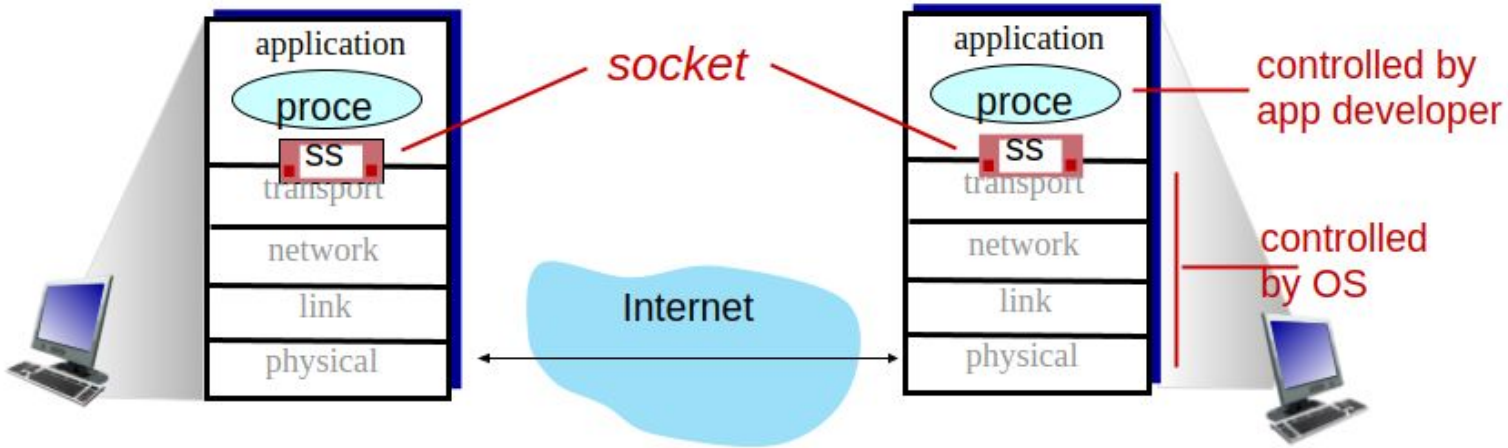
client process: process that initiates communication

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

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” r 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	TCP or UDP
streaming audio/video	proprietary HTTP [RFC 7230], DASH	TCP UDP or TCP
interactive games	WOW, FPS (proprietary)	

Tornando o TCP seguro

TCP & UDP

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

Transport Layer Security - 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:

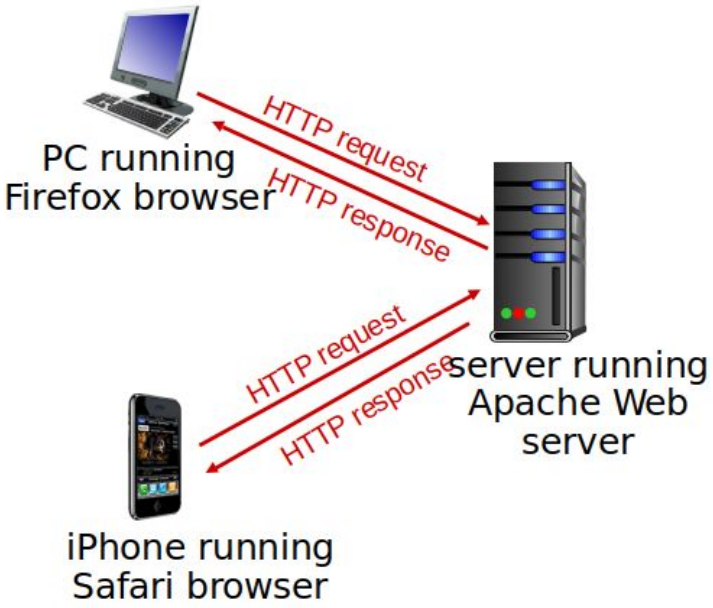
`www.someschool.edu/someDept/pic.gif`

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

Nota

Protocolos que mantêm "estado" são complexos!

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

Conexões HTTP

HTTP não persistente

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/algumDepartmento/inicial.index

(contém texto, referências a 10 imagens jpeg)

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

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

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

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

tempo

Exemplo de HTTP não persistente (cont.)

4. servidor http encerra conexão TCP .

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

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

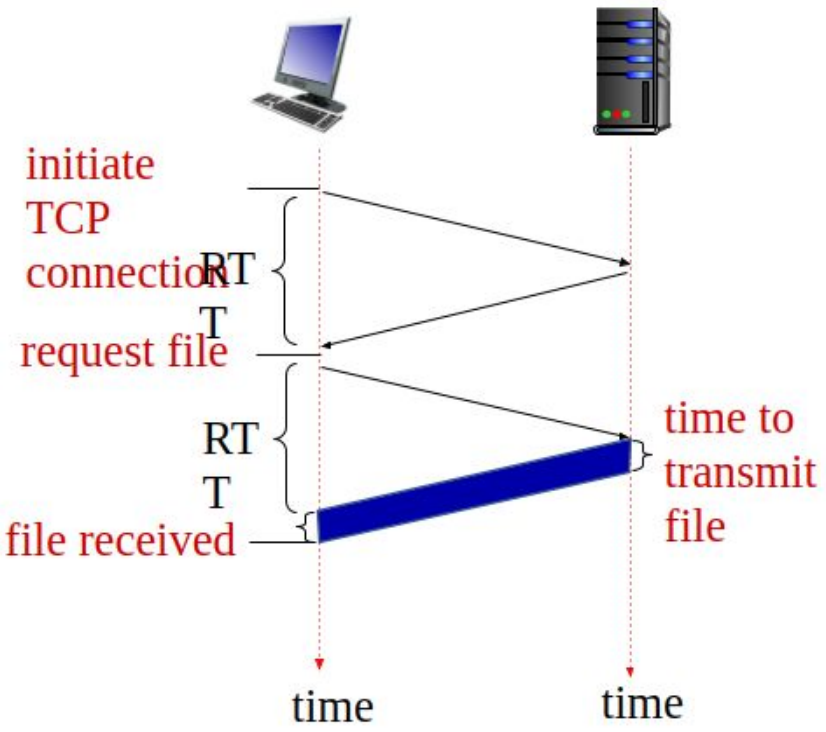
tempo

Modelagem do tempo de resposta

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

Tempo de resposta:

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



Non-persistent HTTP response time = 2RTT+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)

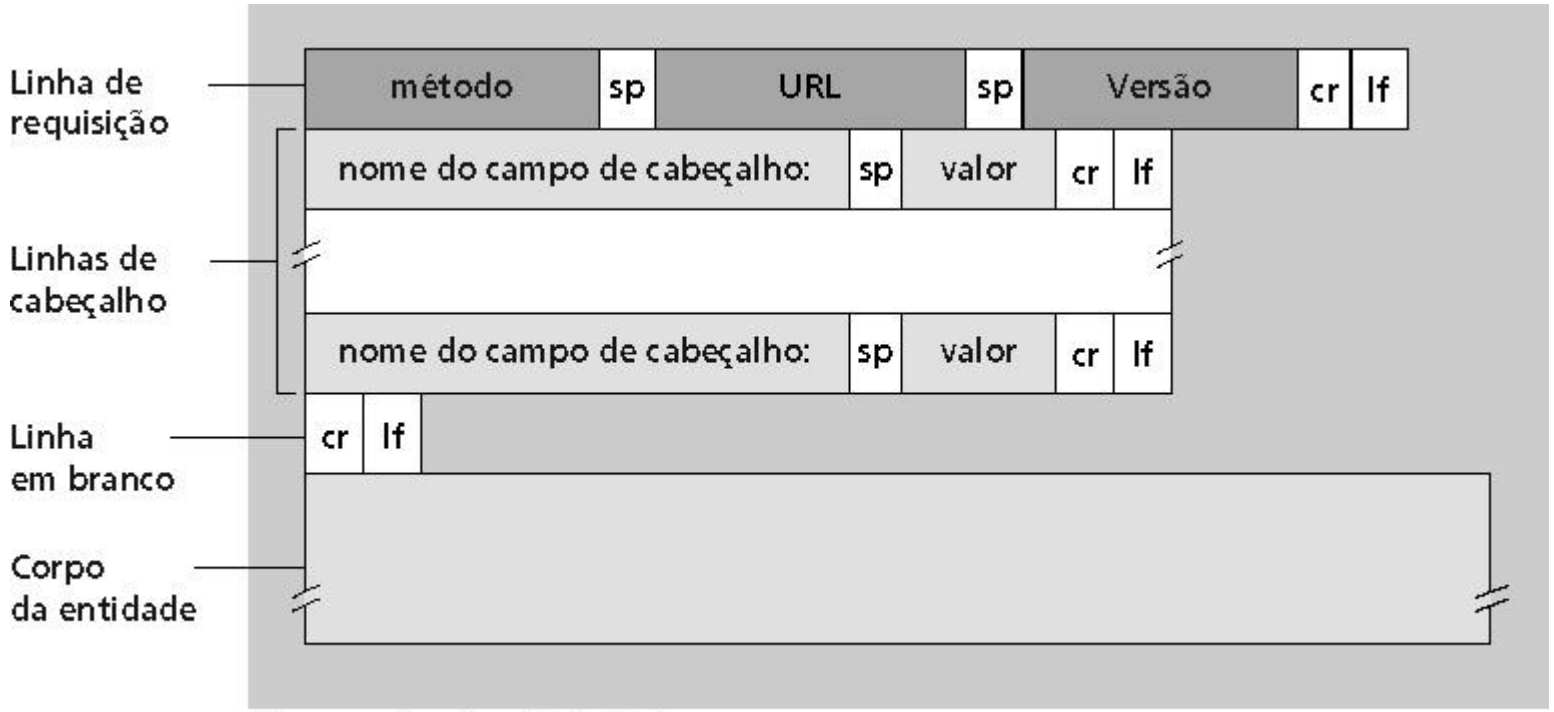
linha da requisição (comandos GET, POST, HEAD)

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



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

Enviando conteúdo de formulário

Método POST :

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

Método URL:

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

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

Tipos de métodos

HTTP/1.0

- GET
- POST
- HEAD

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

HTTP/1.1

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

Mensagem de resposta HTTP

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

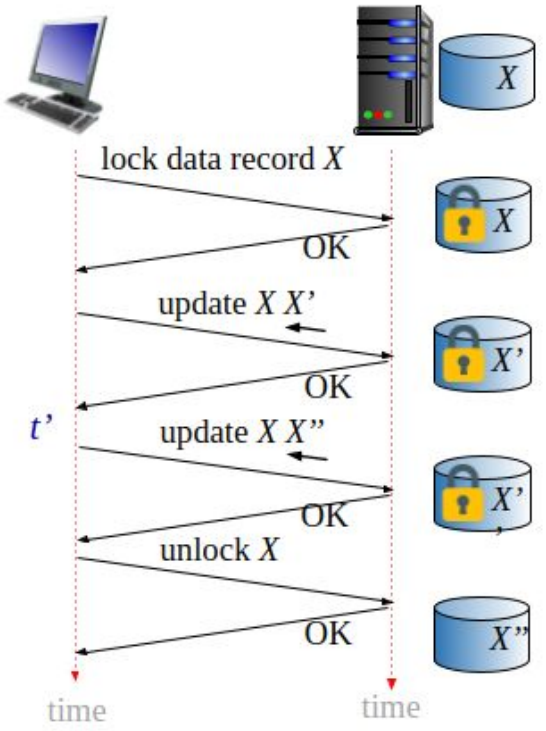
Cookies: manutenção do “estado” da conexão

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.

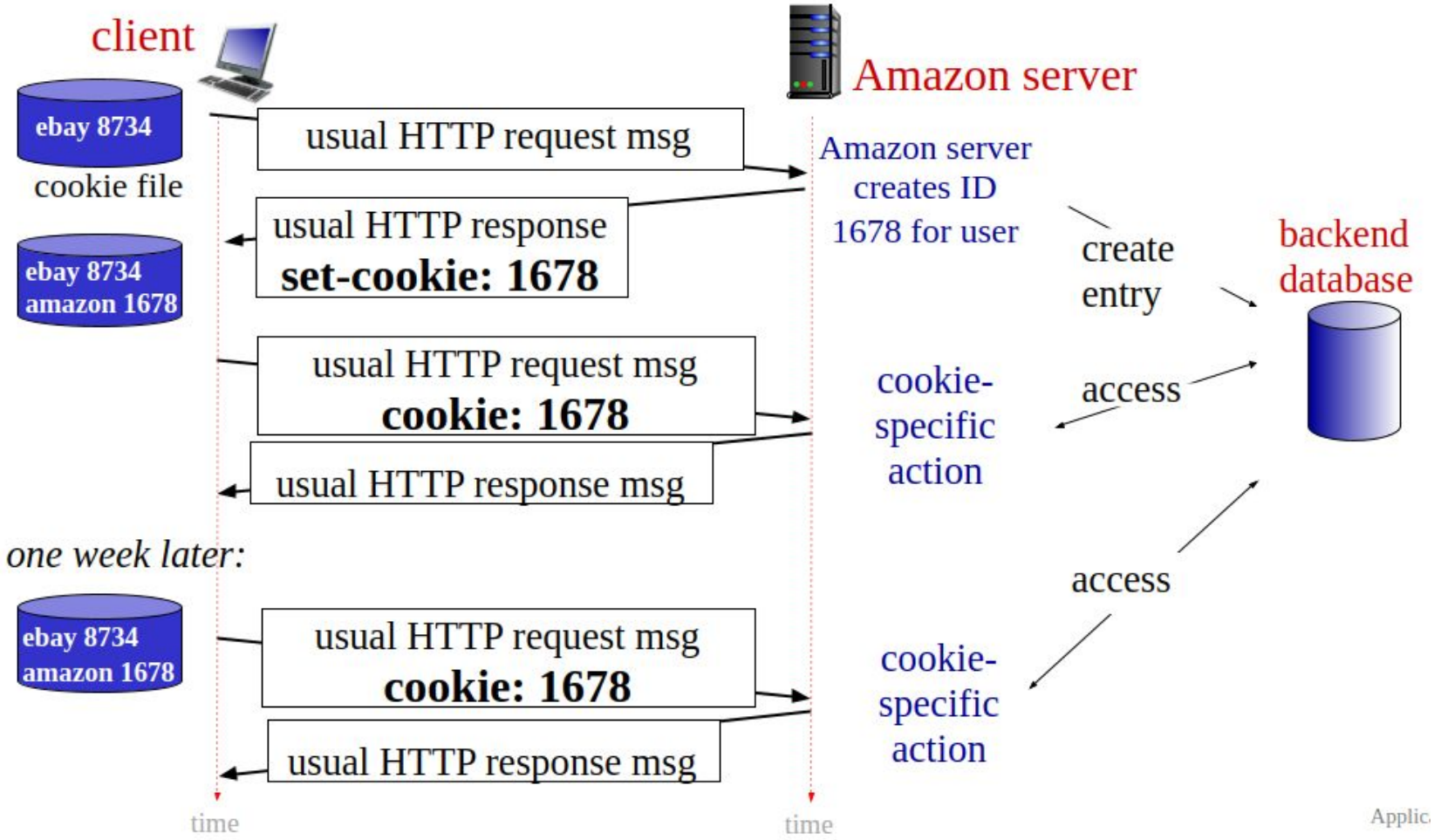
Quatro componentes:

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:

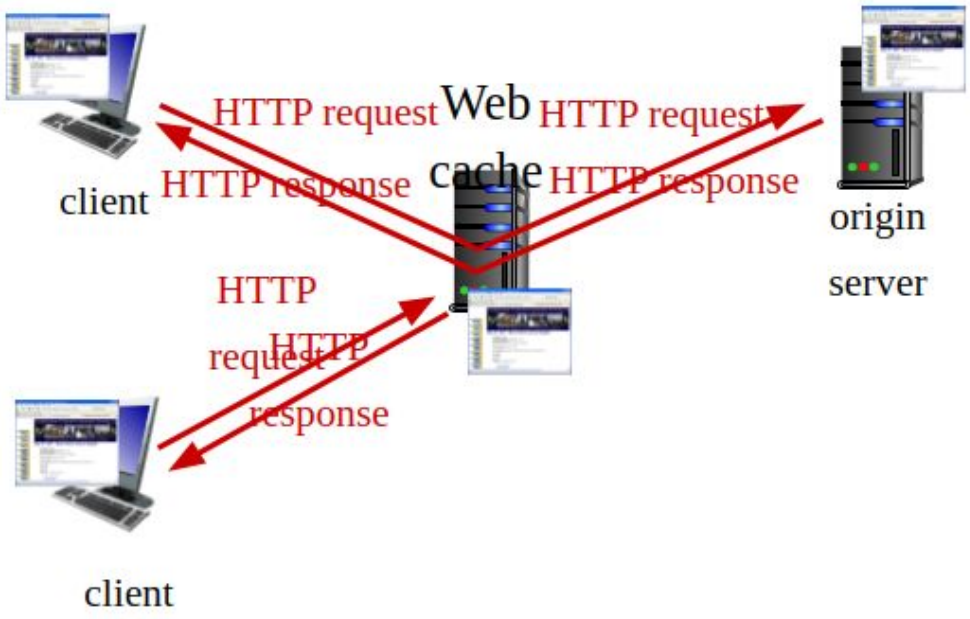
Qual é a função dos cookies na web?

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

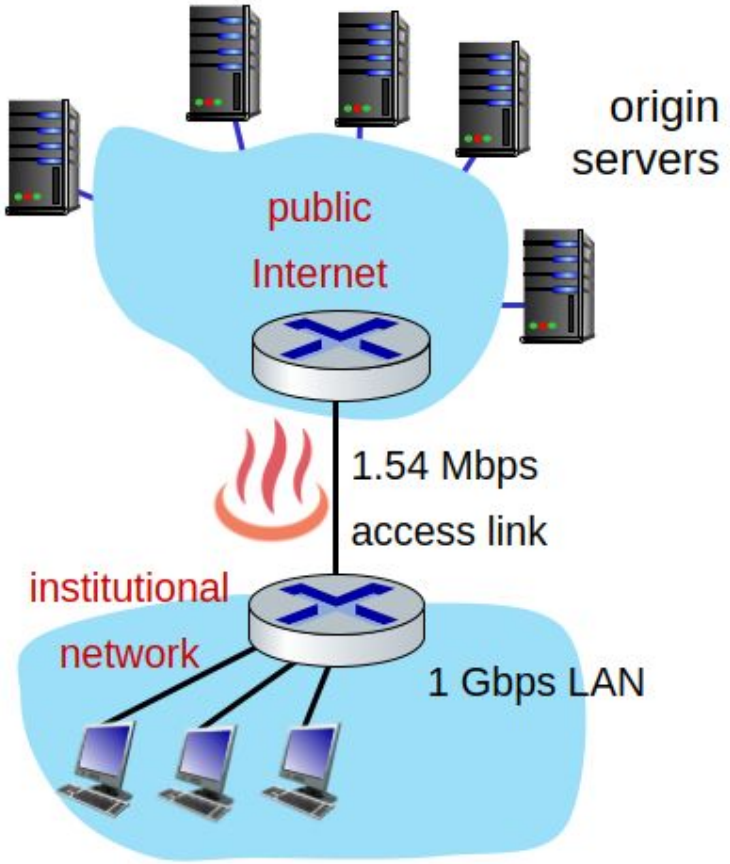
Exemplo de cache (1)

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 + microssegundos



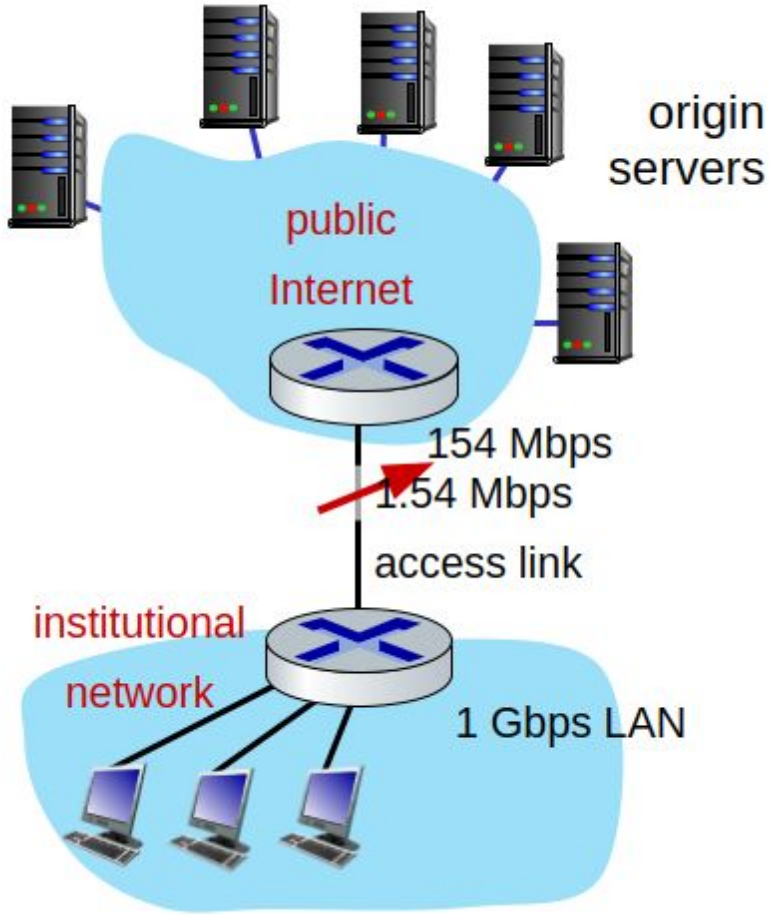
Opção 1: Adquirir um Link de acesso mais veloz

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 + microssegundos
- Frequentemente este é uma ampliação cara



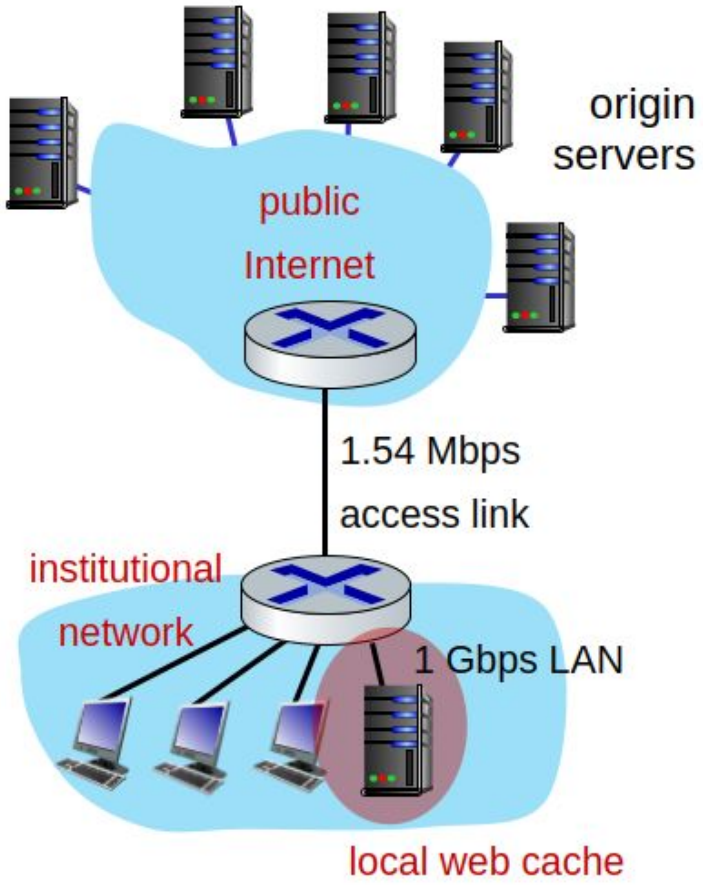
Opção 2: Instalação de um cache

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*2 seg + 0,6*0,01 segs + msegs < 1,3 segs



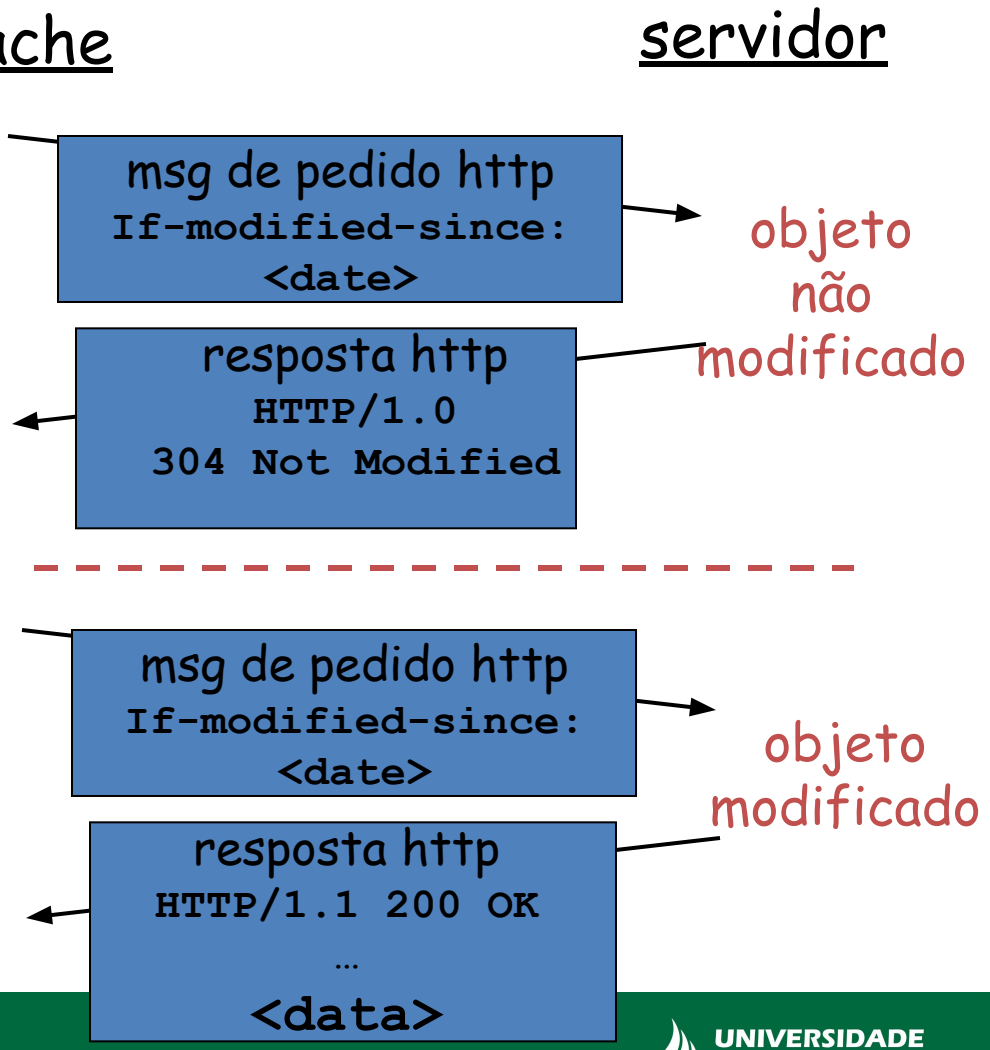
GET condicional

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

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

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

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



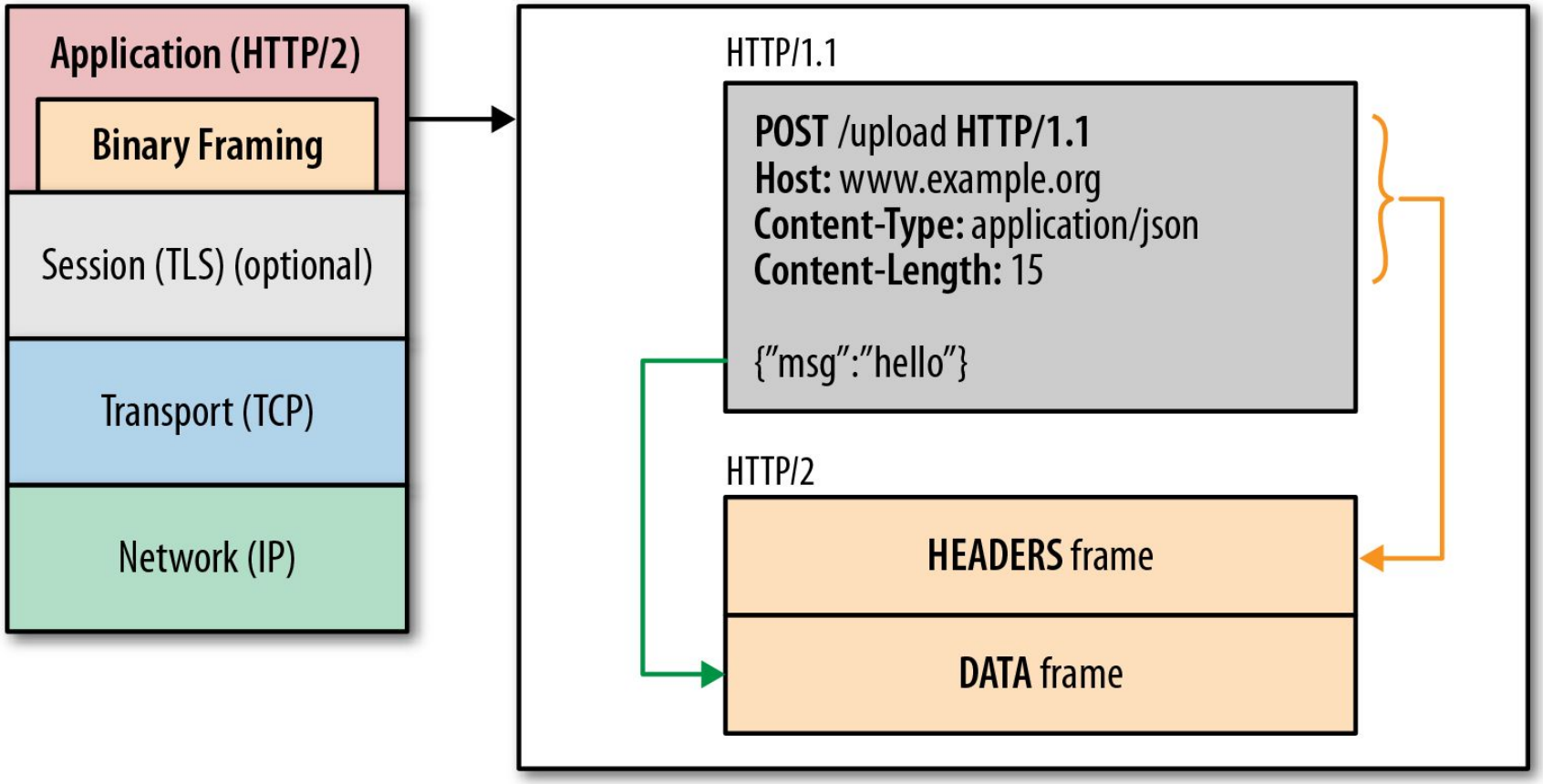
HTTP/2

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

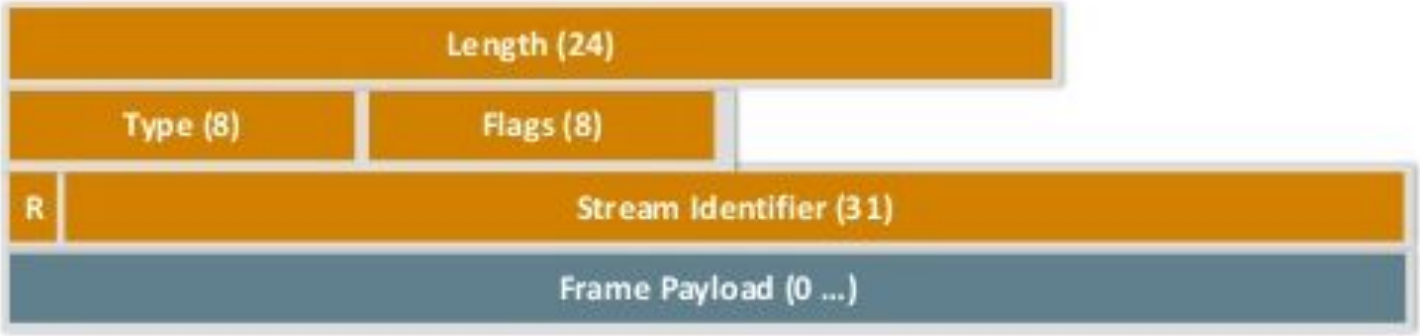
HTTP/2: Diferenças do HTTP 1.1

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

HTTP/2: Transporte Binário



HTTP/2: Quadros



Tipos:

HEADERS, DATA, PRIORITY, RST_STREAM, SETTINGS, PUSH_PROMISE, PING, GOAWAY, WINDOW_UPDATE, CONTINUATION

HTTP/2: Multiplexação