PTC 3450 - Aula 06

2.2 A *Web* e o HTTP

(Kurose, p. 73 - 83)
(Peterson, p. 425 - 444)
24/03/2017

Capítulo 2: conteúdo

2.1 Princípios de aplicativos de rede

2.2 *Web* e HTTP

2.3 Correio eletrônico

- SMTP, POP3, IMAP

2.4 DNS

2.5 Aplicativos P2P

2.6 *Streaming* de vídeo e redes de distribuição de conteúdo

2.7 Programando *socket* com UDP e TCP

Web e HTTP

Primeiro, uma revisão…

- *página web* consiste de *objetos (arquivos)*
- objeto pode ser arquivo HTML, imagem JPEG, *applet* Java, arquivo de áudio,…
- página *web* consiste de *arquivo HTML base* que inclui *diversos objetos referenciados*
- cada objeto é endereçável por uma *URL (Uniform Resource Locator),* e.g.,

www.lcs.poli.usp.br/~marcio/index_arquivos/image002.jpg

nome do *host* | local do objeto

Visão geral do HTTP

HTTP: *HyperText Transfer Protocol*

- RFC 1945 (HTTP 1.0 - 1996)
- RFC 2616 (HTTP 1.1 - 1999)
- RFC 7540 (HTTP 2.0 - 2015)

- protocolo da camada de aplicação da *Web*

- modelo cliente/servidor
  - *cliente:* navegador que pede, recebe (usando protocolo HTTP) e "apresenta" objetos *Web* (Microsoft Edge, Firefox, Chrome)

  - *servidor:* servidor *Web* envia (usando protocolo HTTP) objetos em resposta a requisições (Apache, Microsoft Internet Information Server)

Visão geral do HTTP (continuação)

usa TCP:

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

HTTP é "sem memória"

- servidor não mantém informação sobre pedidos anteriores do cliente

nota

protocolos que mantêm "memória" são complexos!

- história passada (estado) precisa ser mantido
- se cliente/servidor cai, suas visões do "estado" podem ser inconsistentes, precisam ser reconciliadas

Conexões HTTP

HTTP não persistente

- no máximo um objeto enviado sobre uma conexão TCP
  - conexão então fechada
- fazer *download* de múltiplos objetos requer múltiplas conexões

HTTP persistente

- múltiplos objetos podem ser enviados sobre única conexão TCP entre cliente, servidor
- padrão

HTTP não-persistente

suponha que usuário digita URL:
http://www.lcs.poli.usp.br/~marcio/index.htm

(contém texto e referências a 10 imagens jpeg)

1a. HTTP cliente inicia conexão TCP ao (processo) servidor HTTP em www.lcs.poli.usp.br na porta 80

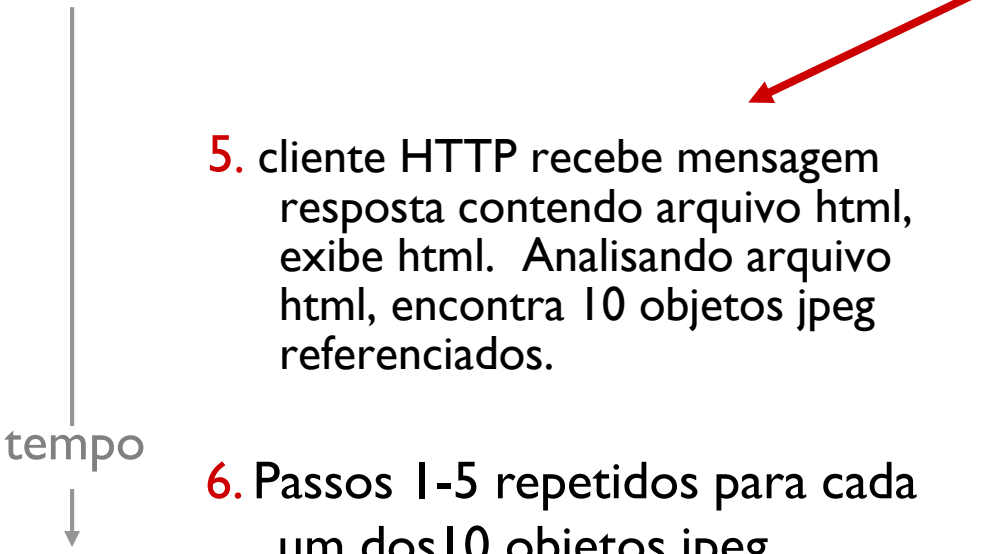
1b. servidor HTTP no *host* www.lcs.poli.usp.br espera por conexão TCP na porta 80. "aceita" conexão, notificando cliente

2. cliente HTTP envia *mensagem pedido* HTTP (contendo URL) para o *socket* de conexão TCP. Mensagem indica que o cliente quer objeto /~marcio/index.htm

3. servidor HTTP recebe mensagem pedido, forma *mensagem resposta* contendo objeto solicitado, e envia mensagem pelo seu *socket*

tempo

HTTP não-persistente (cont.)



4. servidor HTTP fecha conexão TCP.

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

6. Passos 1-5 repetidos para cada um dos10 objetos jpeg

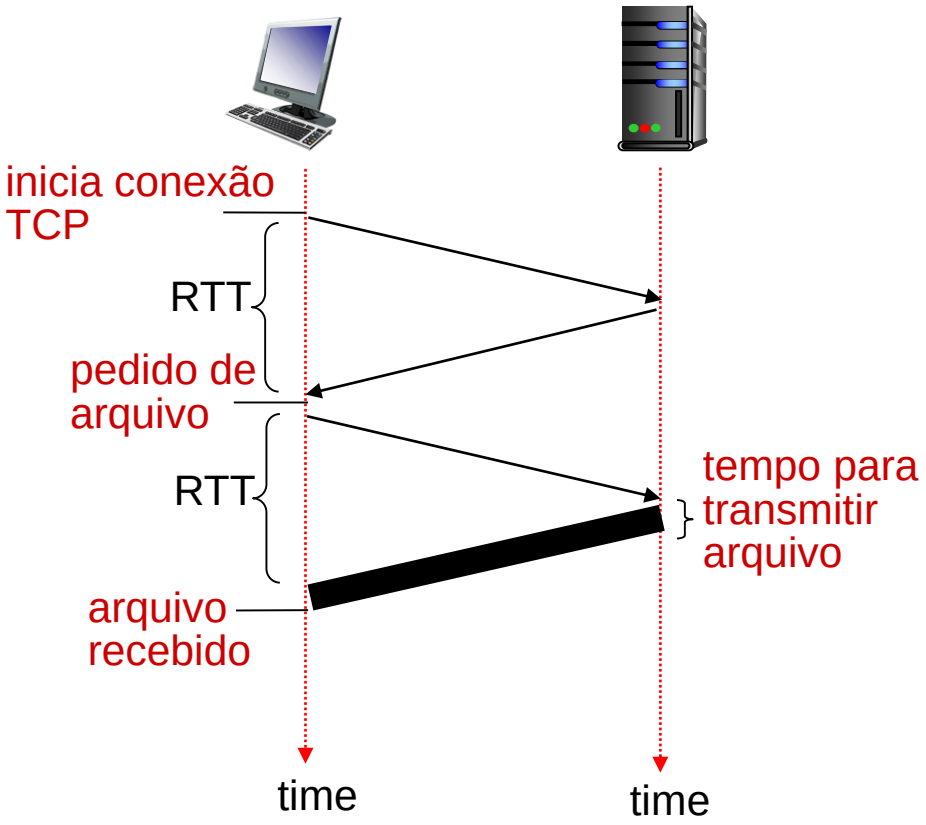
HTTP não-persistente: tempo de resposta

RTT (Round-Trip Time) : tempo para pequeno pacote viajar do cliente ao servidor e voltar

tempo de resposta HTTP:

- um RTT para iniciar conexão TCP
- um RTT para pedido HTTP e primeiros bytes da resposta HTTP retornar
- tempo de transmissão do arquivo
- tempo de resposta para HTTP não-persistente =

  2RTT+ tempo de transmissão do arquivo



HTTP Persistente

Problemas do HTTP não persistente :

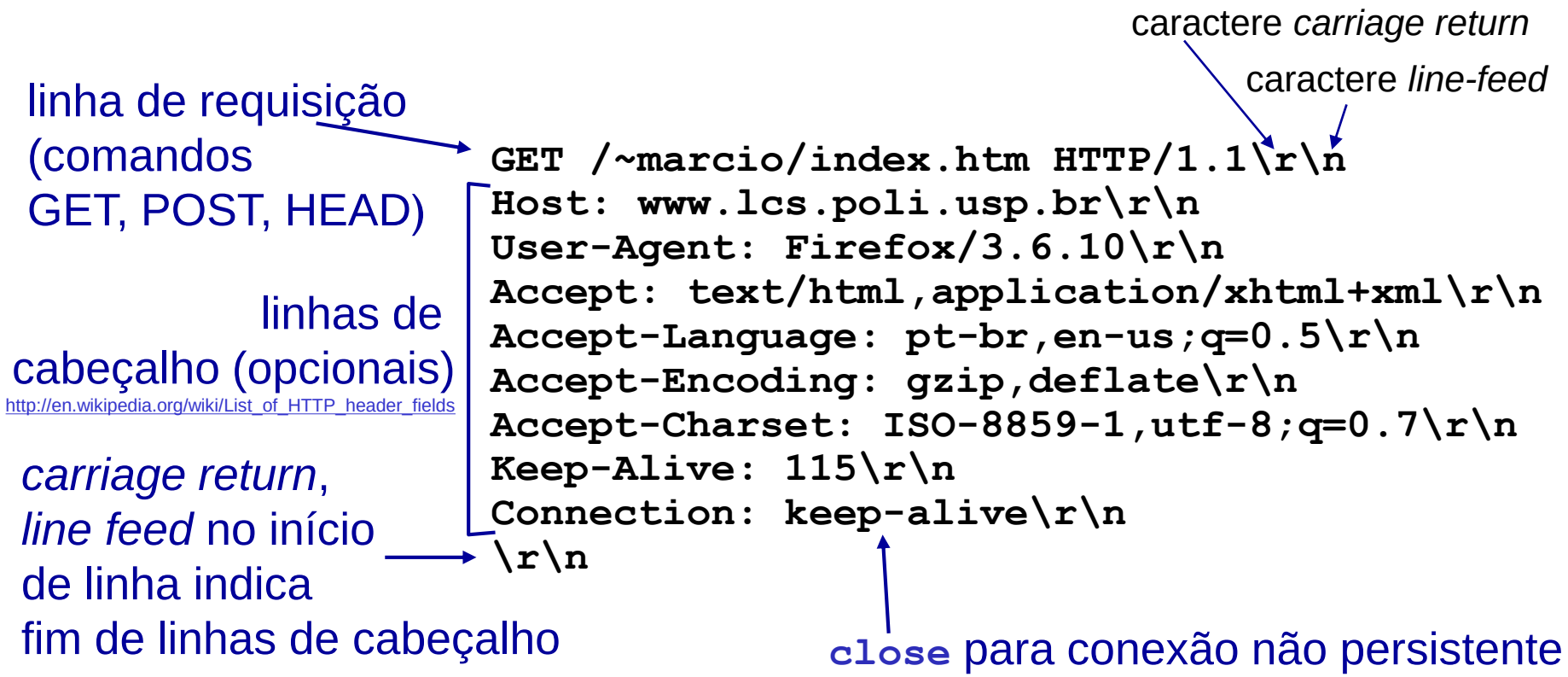
- requer 2 RTTs por objeto
- Sistema operacional precisa gerenciar *cada* conexãoTCP
- navegadores muitas vezes abrem conexões TCP paralelas para baixar objetos referenciados

HTTP persistente:

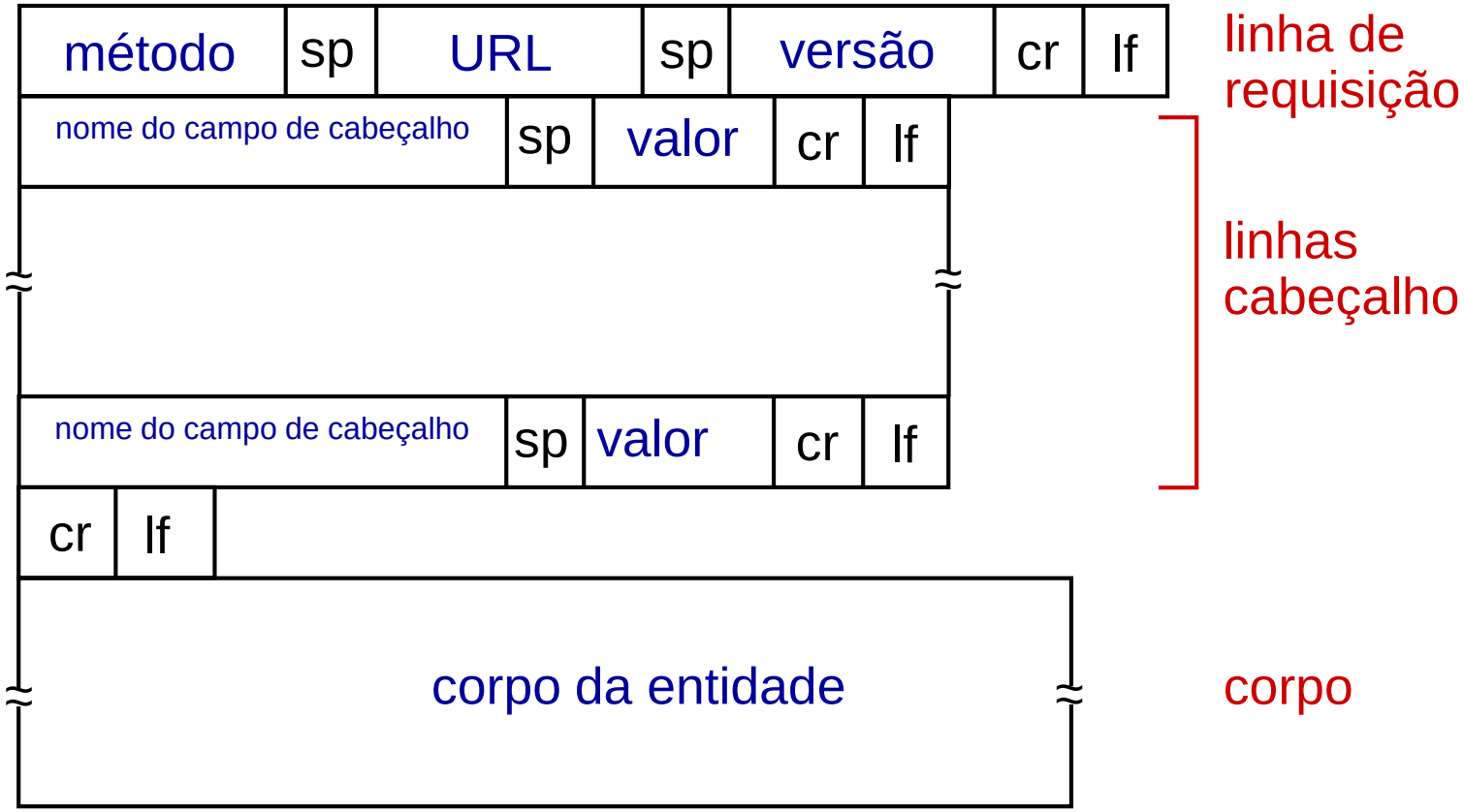
- servidor deixa conexão aberta depois de enviar resposta
- mensagens HTTP subsequentes entre mesmo cliente/servidor enviadas sobre a conexão aberta
- cliente envia pedido assim que encontra objeto referenciado
- perto de 1 RTT para todos os objetos referenciados

Mensagem pedido HTTP

- 2 tipos de mensagens HTTP: *pedido (request), resposta*
- Mensagem pedido HTTP:
  - ASCII (formato que permite leitura por humanos)



Mensagem pedido HTTP: formato geral



Upload de entrada de formulário

método POST:

- páginas *web* muitas vezes incluem formulário de entrada
- *upload* da entrada para servidor é feita no corpo da mensagem

método URL :

- usa método GET
- *upload* da entrada é feito no campo URL da linha de pedido:

`http://www.imdb.com/find?ref_=nv_sr_fn&q=casablanca&s=all`

dados de entrada

Tipos de métodos

HTTP/1.0:

- GET
- POST
- HEAD
  - pede ao servidor que deixe objeto pedido fora da mensagem

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - faz *upload* do arquivo no corpo da mensagem no caminho especificado no campo URL
- DELETE
  - apaga o arquivo especificado no campo URL

Mensagem resposta HTTP

linha de estado (código e frase de estado do protocolo)

linhas de cabeçalho

dados, e.g., arquivo HTML requisitado

```
HTTP/1.1 200 OK\r\n
Date: Tue, 25 Feb 2014 18:24:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 18 Feb 2014 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 estado da resposta HTTP

- Código de estado aparece na 1a linha da mensagem resposta servidor-cliente
- Alguns códigos exemplos:

200 OK

- atendido com sucesso, objeto pedido mais para frente na msg

301 Moved Permanently

- objeto pedido foi movido, nova localização especificada mais a frente nessa msg (Location:)

400 Bad Request

- mensagem pedido não entendida pelo servidor

404 Not Found

- documento pedido não encontrado nesse servidor

505 HTTP Version Not Supported

Experimentando o HTTP (lado cliente)

1. Acesse um servidor *web* usando *Telnet*:

```
telnet
set localecho
o www.lcs.poli.usp.br 80
```

No Windows, abre conexão TCP para porta 80 (porta servidor HTTP padrão) em www.lcs.poli.usp.br. Qualquer coisa digitada é enviada pela porta 80 em www.lcs.poli.usp.br

2. digite um pedido GET HTTP:

```
GET /~marcio/branco.htm HTTP/1.1
Host: www.lcs.poli.usp.br
```

digitando essa mensagem (aperte ENTER duas vezes), você envia um pedido GET simples (mas completo) ao servidor HTTP

3. veja a resposta enviada pelo servidor HTTP!

(ou use *Wireshark* para examinar pedidos/respostas HTTP capturados)

(Kurose, p. 125) Considere o seguinte *string* de caracteres ASCII que foram capturados pelo *Wireshark* quando o navegador enviou uma mensagem HTTP GET (isto é, esse é o conteúdo real da mensagem HTTP GET). Os caracteres *<cr><lf>* são caracteres *carriage return* e *line-feed* . Responda as seguintes questões, indicando onde na mensagem HTTP GET abaixo você encontra a sua resposta.

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (
Windows;U; Windows NT 5.1; en-US; rv:1.7.2) Gec
ko/20040804 Netscape/7.2 (ax) <cr><lf>Accept:ex
t/xml, application/xml, application/xhtml+xml, text
/html;q=0.9, text/plain;q=0.8,image/png,*/*;q=0.5
<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-
Encoding: zip,deflate<cr><lf>Accept-Charset: ISO
-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr>
<lf>Connection:keep-alive<cr><lf><cr><lf>
```

(a) Qual o URL do documento requisitado pelo navegador?

(b) Qual a versão de HTTP o navegador está rodando?

(c) O navegador requisitou uma conexão persistente ou não persistente?

(d) Qual é o endereço IP do *host* no qual o navegador está rodando?

(e) Que tipo de navegador iniciou a mensagem? Por que é necessário o tipo de navegador numa mensagem de pedido HTTP?

Estado usuário-servidor : cookies (RFC6265)

muitas páginas *Web* usam *cookies*

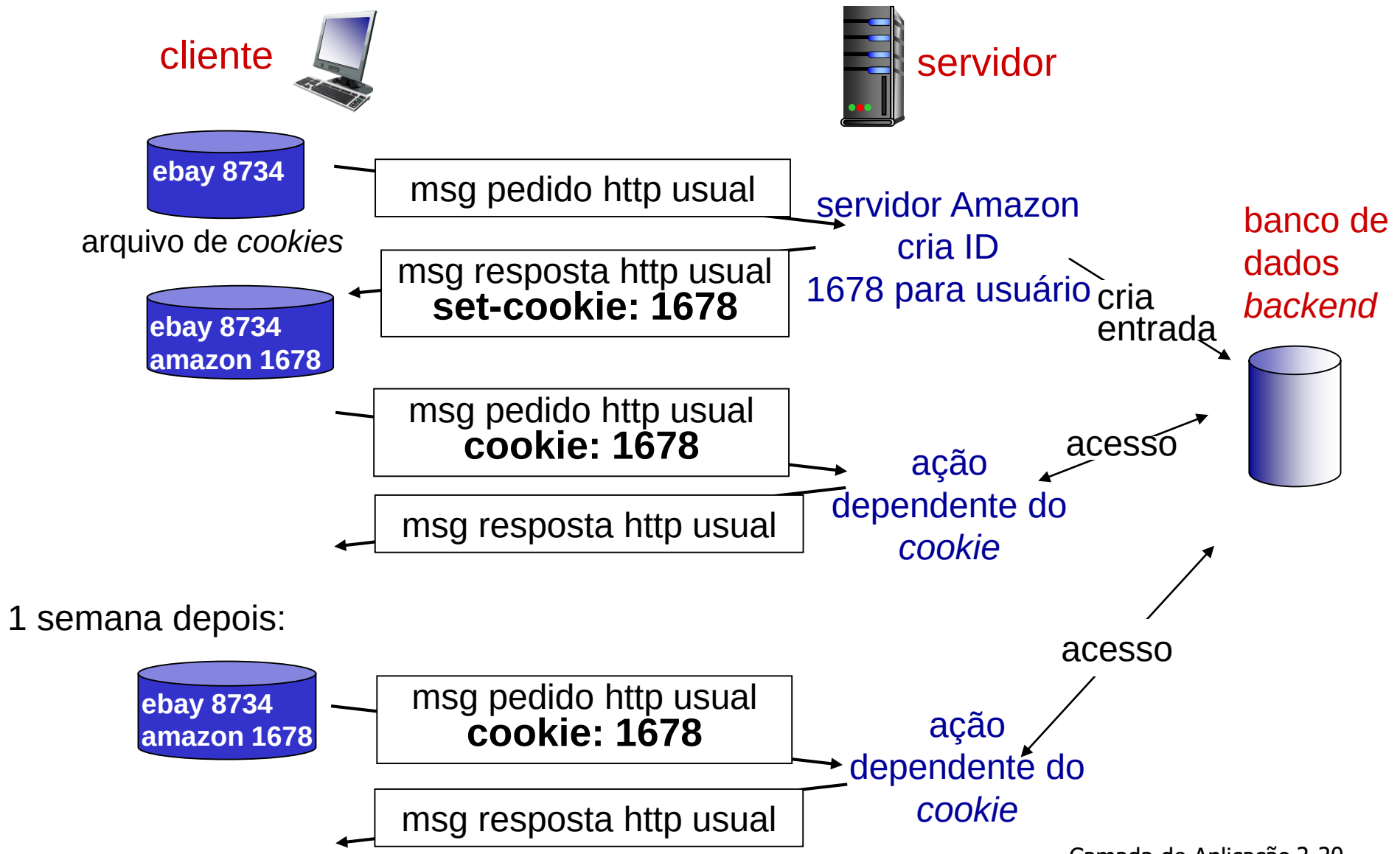
4 componentes:

1) linha de cabeçalho *cookie* em mensagem *resposta* HTTP
2) linha cabeçalho *cookie* na próxima mensagem pedido HTTP
3) arquivo de *cookies* mantido no *host* do usuário, gerenciado pelo navegador do usuário
4) banco de dados *back-end* na página *Web*

exemplo:

- Beatriz sempre acessa a Internet de seu PC
- visita *site* de *e-commerce* específica pela primeira vez
- quando pedido HTTP inicial chega ao *site*, são criados:
  - ID única
  - entrada no banco de dados *back end* para ID

Cookies: mantendo o "estado" (cont.)



Cookies (continuação)

cookies podem ser usados para:

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

nota

cookies e privacidade:

- *cookies* permitem que sites aprendam muito sobre você
- principalmente se você forneceu nome e *e-mail* também

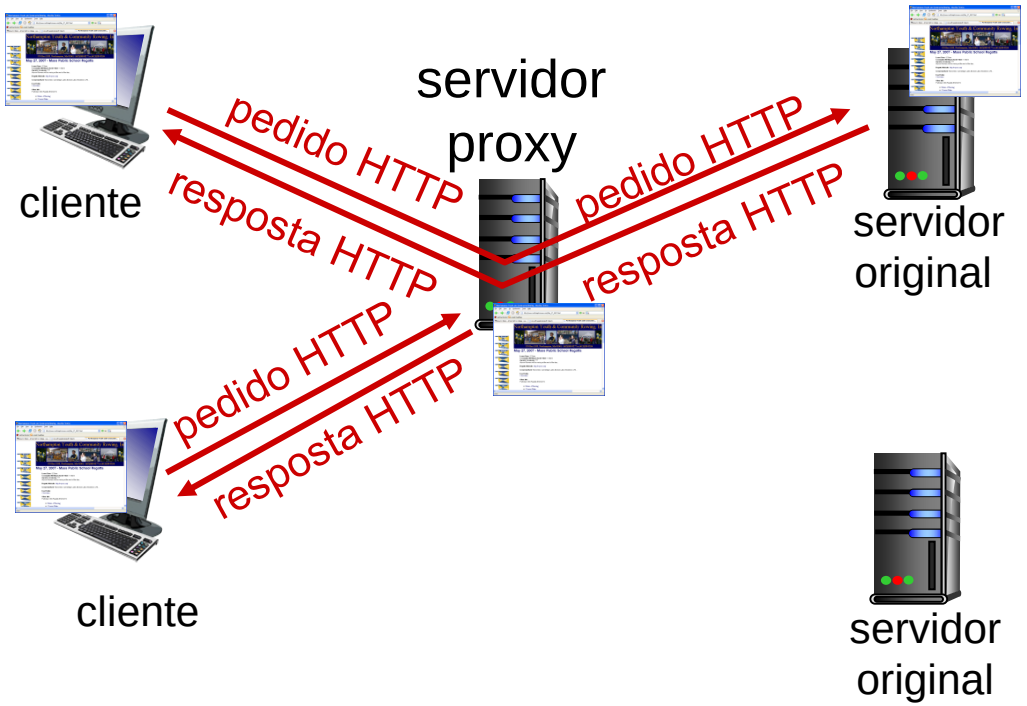
como manter "estado":

- protocolos nas extremidades: mantêm estado no transmissor/receptor sobre múltiplas comunicações
- *cookies*: mensagens http carregam estado

Caches *Web* (servidores *proxy*)

objetivo: atender pedido cliente sem envolver servidor original

- usuário configura navegador: acessos Web via *cache*
- navegador envia todos pedidos HTTP para *cache*
  - objetos no *cache*: *cache* retorna objeto
  - se não *cache* requer objetos do *servidor* original, então retorna objeto a cliente



Mais sobre *Web caching*

- *cache* atua tanto como cliente quanto servidor
  - servidor para pedido do cliente
  - cliente para servidor original
- tipicamente *cache* é instalado por ISPs (universidade, empresa, ISP residenciais)

por que Web caching?

- reduzir tempo de respota para pedido de cliente
- reduzir tráfego no enlace de acesso da instituição
- Internet tem muitos *caches*: permite provedores de conteúdo "pobres" fornecer conteúdo eficientemente (assim como compartilhamento P2P)

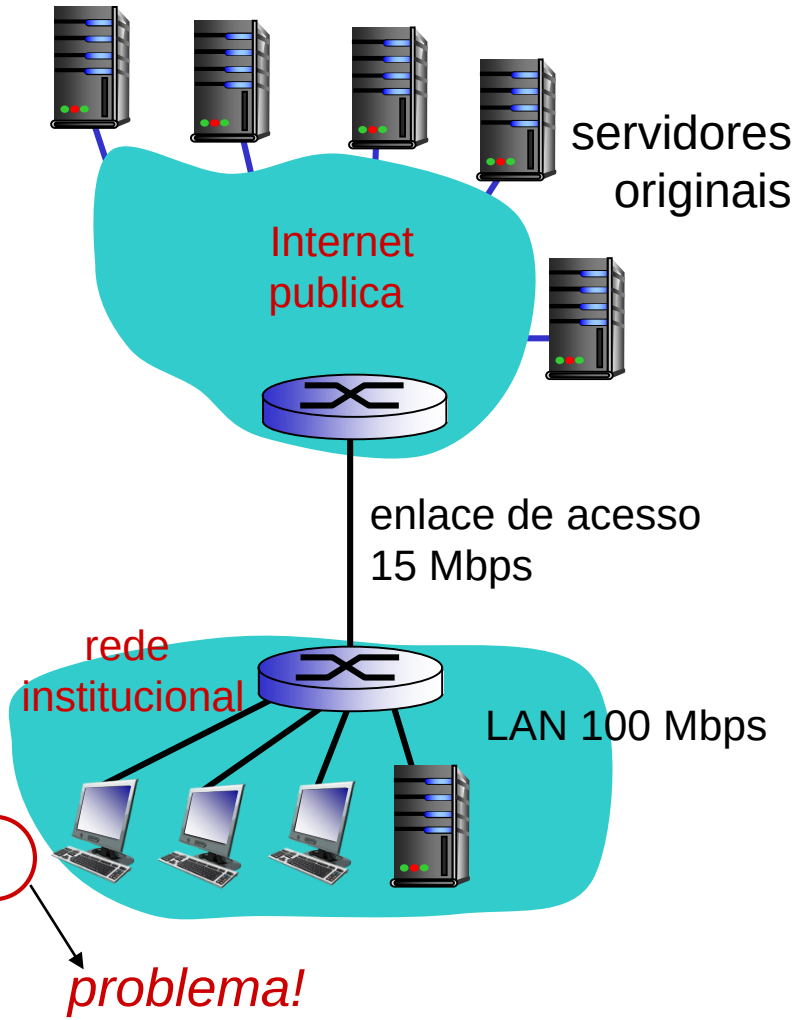
Exemplo de *caching*:

hipoteses:

- tamanho médio de objeto: 1 Mbit
- taxa média de pedidos dos navegadores aos servidores originais: 15/s
- taxa de dados média para os navegadores: 15 Mbps
- RTT do roteador da Internet mais próximo a qualquer servidor original (atraso internet): 2 s
- capacidade do enlace de acesso: 15 Mbps

consequências:

- intensidade de tráfego LAN: (15 pedidos/s)*(1 Mbit/pedido)/(100 Mbps) = 0.15
- intensidade de tráfego enlace de acesso = 1
- latência total = atraso Internet + atraso do acesso + atraso LAN
  = 2 s + minutos + ms



Exemplo de *caching*: melhorando enlace de acesso

hipoteses:

- tamanho médio de objeto: 1 Mbits
- taxa média de pedidos dos navegadores aos servidores originais: 15/s
- taxa de dados média para os navegadores: 15 Mbps
- RTT do roteador da Internet mais próximo a qualquer servidor original (atraso internet): 2 s
- capacidade do enlace de acesso: ~~15 Mbps~~ 100 Mbps

consequências:

- intensidade de tráfego LAN: (15 pedidos/s)*(1 Mbit/pedido)/(100 Mbps) = 0.15
- intensidade de tráfego enlace de acesso = 0.15
- latência total = atraso Internet + atraso do acesso + atraso LAN
  = 2 s + ~~minutos~~ ms
  ms

Custo: aumento da velocidade do enlace de acessor (caro!)

Exemplo de *caching*: instalando *cache* local

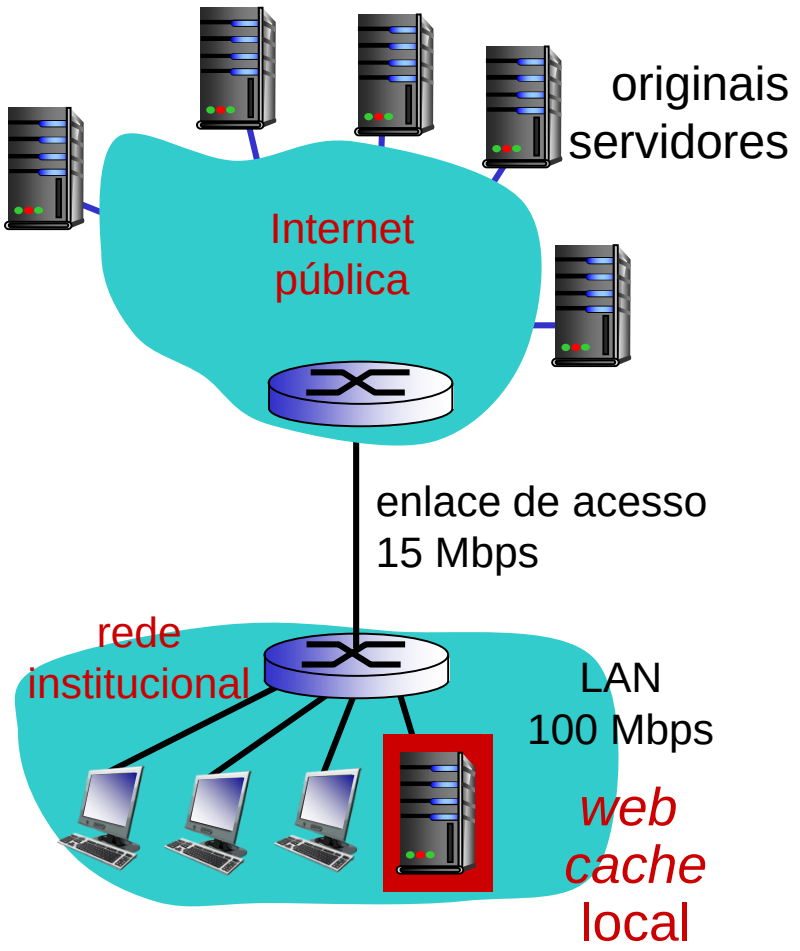
hipóteses:

- tamanho médio de objeto: 1 Mbits
- taxa média de pedidos dos navegadores aos servidores originais: 15/s
- taxa de dados média para os navegadores: 15 Mbps
- RTT do roteador da Internet mais próximo a qualquer servidor original (atraso internet): 2 s
- capacidade do enlace de acesso: 15 Mbps

consequências:

- intensidade de tráfego LAN : 15%
- intensidade de tráfego enlace de acesso = ?
- latência total = ?

Como calcular a intensidade de tráfego e a latência?



Custo: web cache (barato!) – software livre em PC

Exemplo de *caching*: instalando *cache* local

Calculando intensidade de tráfego no enlace de acesso e latência com cache:

- supondo *hit rate* do *cache* 0.4 (tipicamente entre 0.2 e 0.7)
  - 40% pedidos atendidos no *cache*,
  - 60% pedidos atendidos na origem
  - intensidade de tráfego:
    - 60% dos pedidos usam enlace de acesso
  - taxa de dados para navegadores sobre o enlace de acesso = 0.6*15 Mbps = 9 Mbps
    - intensidade = 9/15 = .6
  - latência total
    - = 0.6 * (latência desde servidores originais) +0.4 * (latência quando atendidos no *cache*)
    - = 0.6 (~ 2 s) + 0.4 (ms)
    - ~ 1.2 s
    - menor do que com enlace de100 Mbps (e mais barato também!)