

## Aula 5

### Professores:

*Anna Dolejsi Santos (UFF)*

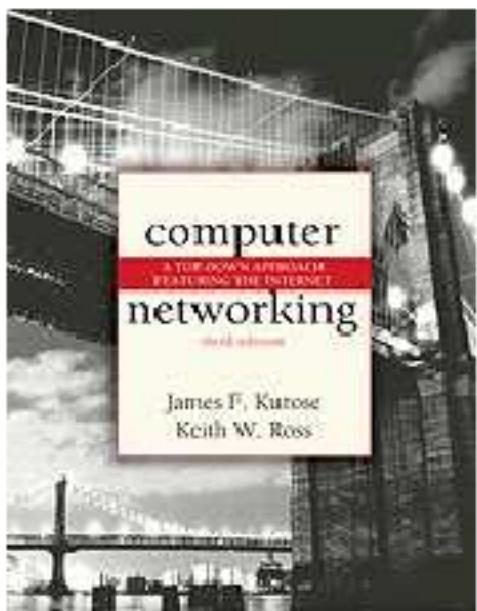
*Célio Vinicius Neves de Albuquerque (UFF)*

## World Wide Web

### Conteúdo:

- Web e HTTP

## Livro Texto



### **REDES DE COMPUTADORES E A INTERNET UMA ABORDAGEM TOP-DOWN**

([www.aw.com/kurose\\_br](http://www.aw.com/kurose_br))

**James F. Kurose e Keith W. Ross**

**Copyright: 2006 - 3a. Edição**

**ISBN: 8588639181**

<http://www.pearson.com.br/>

### **Referências Adicionais:**

**Redes de Computadores**

**Andrew Tanenbaum**

**Editora Campus, 4a. Edição, 2003**

**ISBN: 8535211853**

Obs: As figuras que não têm referências pertencem ao material disponilizado pelo autor do livro texto ou foram produzidas pelo professor desta disciplina.

## Capítulo 2: Roteiro

- 2.1 Princípios dos protocolos da camada de aplicação
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Compartilhamento de arquivos P2P
- 2.7 Programação de Sockets com TCP
- 2.8 Programação de Sockets com UDP

## Web e HTTP

### Primeiro algum jargão

- ❑ Páginas Web consistem de objetos
- ❑ Objeto pode ser um arquivo HTML, uma imagem JPEG, um applet Java, um arquivo de áudio, ...
- ❑ Páginas Web consistem de um arquivo HTML base que inclui vários objetos referenciados
- ❑ Cada objeto é endereçável por uma URL
- ❑ Exemplo de URL:

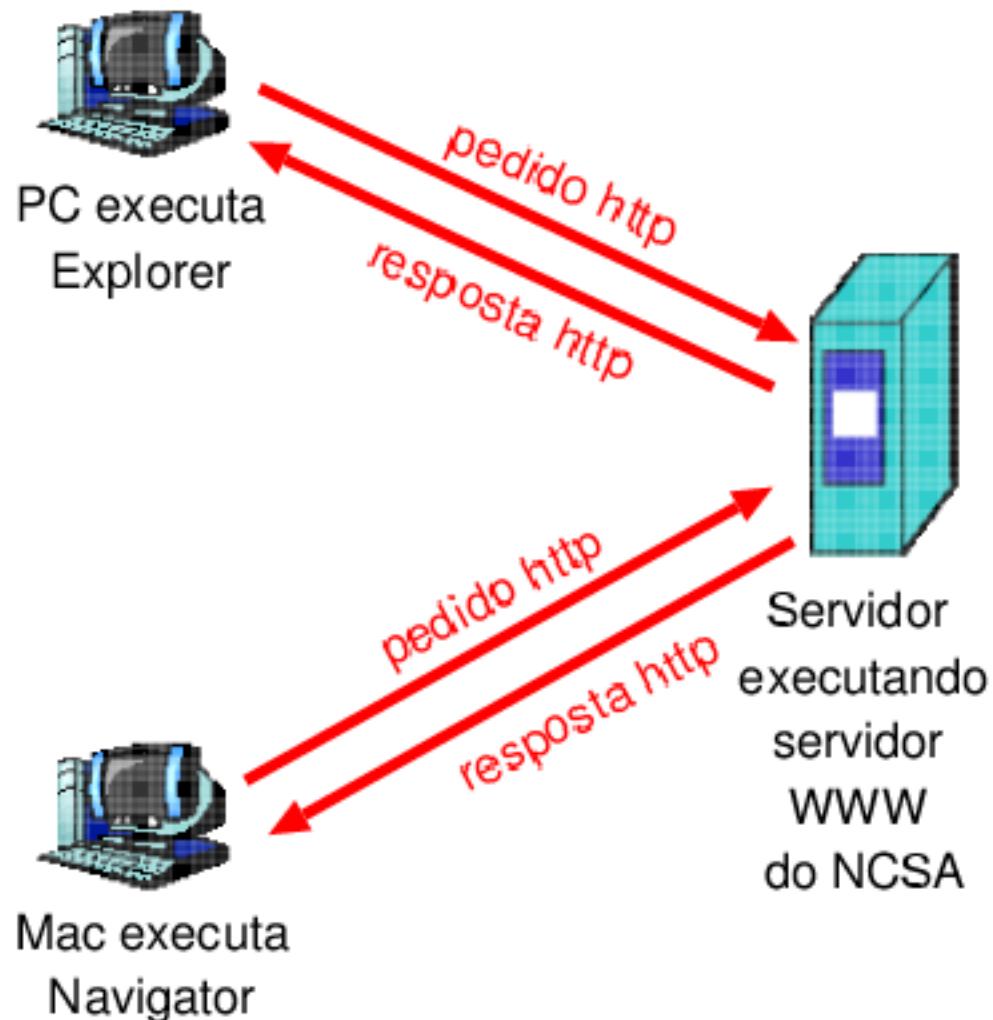
www.someschool.edu/someDept/pic.gif

nome do sistema final                            nome do caminho

## Protocolo HTTP

**HTTP: hypertext transfer protocol**

- protocolo da camada de aplicação da Web
- modelo cliente/servidor
  - **cliente:** browser que pede, recebe, "visualiza" objetos Web
  - **servidor:** servidor Web envia objetos em resposta a pedidos
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



## 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

- No máximo um objeto é enviado numa conexão TCP
- HTTP/1.0 usa o HTTP não persistente

### HTTP persistente

- Múltiplos objetos podem ser enviados sobre uma única conexão TCP entre cliente e servidor
- HTTP/1.1 usa conexões persistentes no seu modo *default*

## 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.

2. cliente http envia *mensagem de pedido* de http (contendo URL)  
através do socket da conexão  
TCP

tempo  
↓

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

3. servidor http recebe mensagem  
de pedido, formula *mensagem de resposta* contendo objeto solicitado  
(`algumDepartmento/inicial.index`),  
envia mensagem via socket

## Exemplo de HTTP não persistente (cont.)

- tempo ↓
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

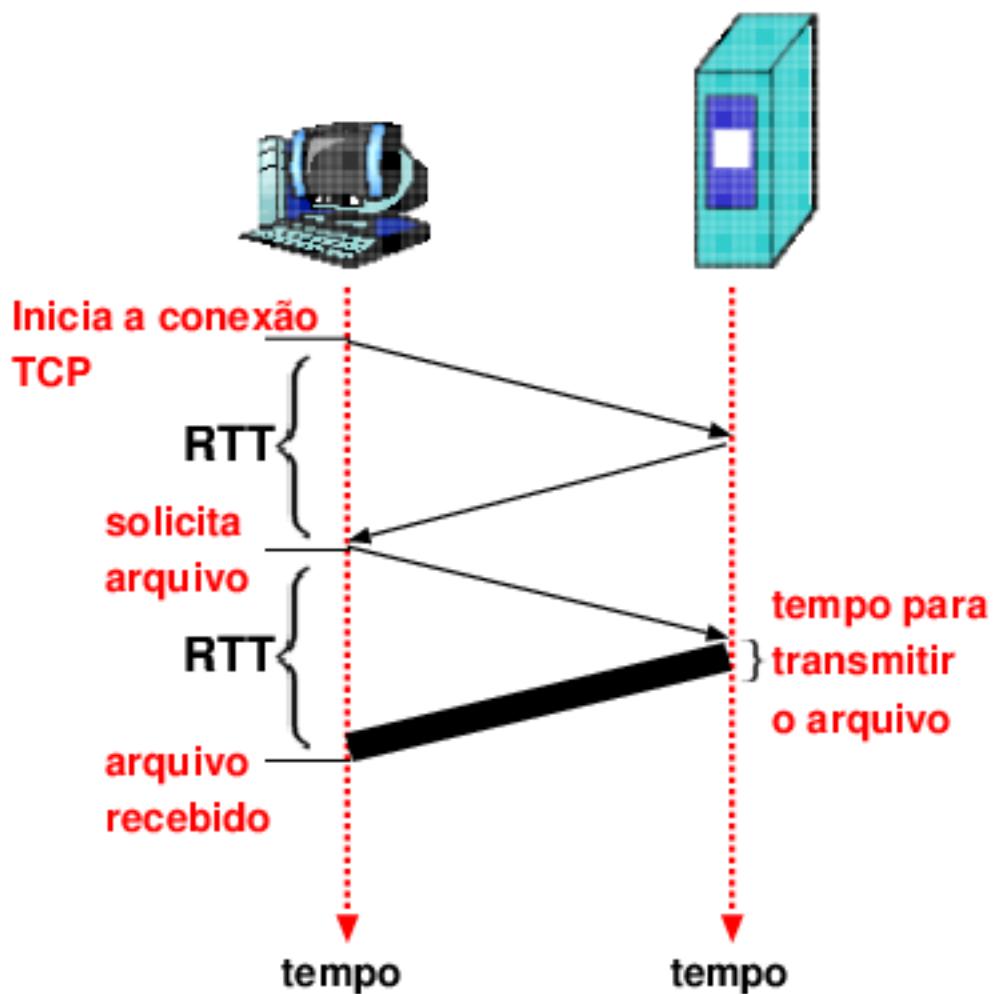
## Modelagem do tempo de resposta

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

$$\text{total} = 2\text{RTT} + \text{tempo de transmissão}$$



## HTTP persistente

### Problemas com o HTTP não persistente:

- requer 2 RTTs para cada objeto
- SO aloca recursos do *host* para cada conexão TCP
- os *browser* freqüentemente abrem conexões TCP paralelas para recuperar os objetos referenciados

### HTTP persistente

- o servidor deixa a conexão aberta após enviar a resposta
- mensagens HTTP seguintes entre o mesmo cliente/servidor são enviadas nesta conexão



### Persistente sem pipelining :

- o cliente envia um novo pedido apenas quando a resposta anterior tiver sido recebida
- um RTT para cada objeto referenciado

### Persistente com pipelining :

- default no HTTP/1.1
- o cliente envia os pedidos logo que encontra um objeto referenciado
- pode ser necessário apenas um RTT para todos os objetos referenciados

## HTTP persistente

### Problemas com o HTTP não persistente:

- requer 2 RTTs para cada objeto
- SO aloca recursos do *host* para cada conexão TCP
- os *browser* freqüentemente abrem conexões TCP paralelas para recuperar os objetos referenciados

### HTTP persistente

- o servidor deixa a conexão aberta após enviar a resposta
- mensagens HTTP seguintes entre o mesmo cliente/servidor são enviadas nesta conexão

### Persistente sem pipelining :

- o cliente envia um novo pedido apenas quando a resposta anterior tiver sido recebida
- um RTT para cada objeto referenciado

### Persistente com pipelining :

- default no HTTP/1.1
- o cliente envia os pedidos logo que encontra um objeto referenciado
- pode ser necessário apenas um RTT para todos os objetos referenciados

## Formato de mensagem HTTP: pedido

- Dois tipos de mensagem HTTP: *pedido, resposta*
- **mensagem de pedido HTTP:**
  - ASCII (formato legível por pessoas)

linha do pedido  
(comandos GET,  
POST, HEAD)

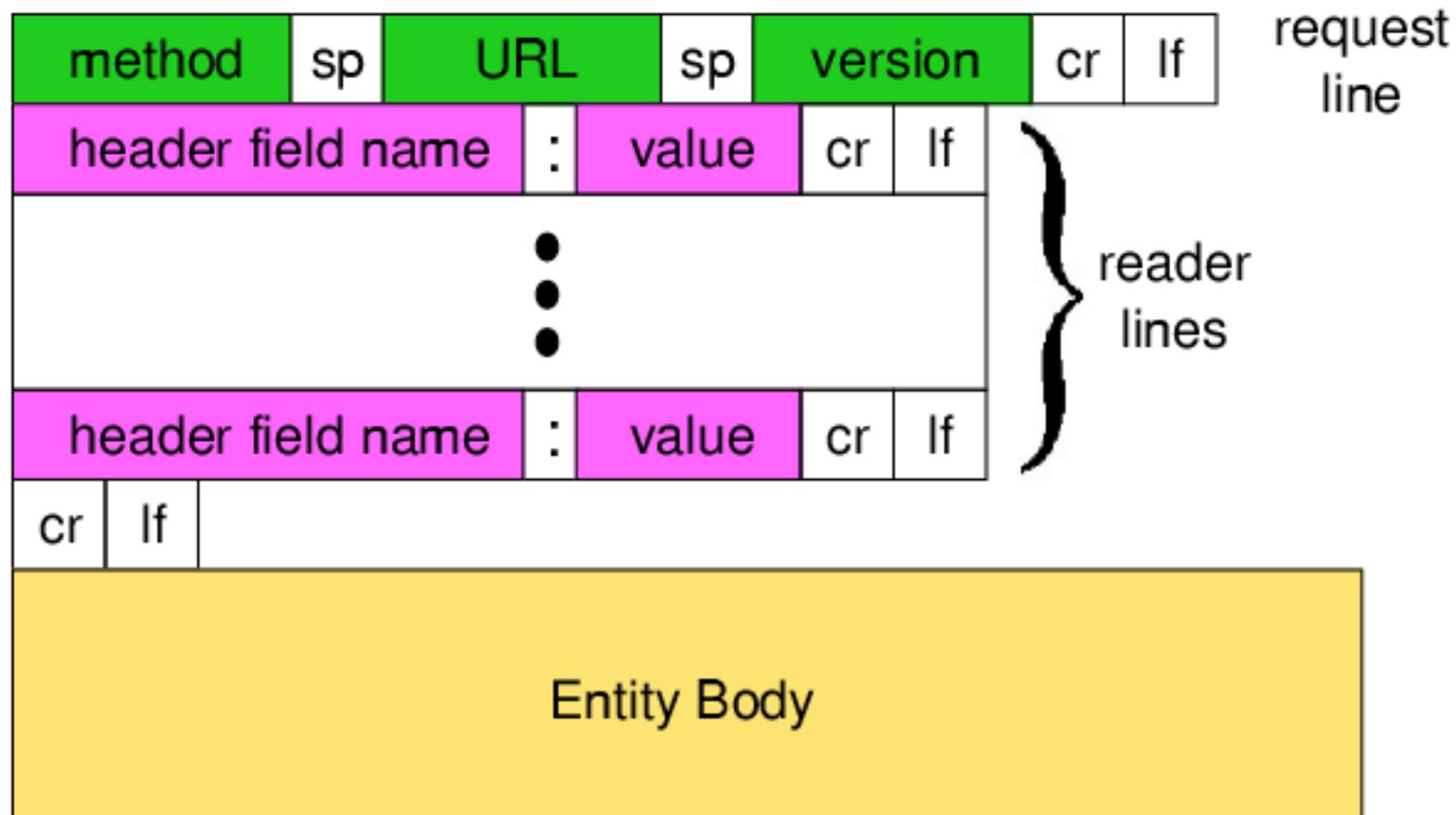
linhas do  
cabeçalho

Carriage return,  
line feed  
indicam fim  
de mensagem

GET /somedir/page.html HTTP/1.0  
Host: www.someschool.edu  
User-agent: Mozilla/4.0  
Connection: close  
Accept-language:fr

(carriage return (CR), line feed(LF)  
pcionais)

## Mensagem de pedido HTTP: formato geral



## Tipos de métodos

### HTTP/1.0

- GET
- POST
- HEAD
- Pede para o servidor não enviar o objeto requerido junto com a resposta (usado p/ debugging)

### 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

## Enviando conteúdo de formulário

### Método POST:

- Conteúdo é enviado para o servidor no corpo da mensagem

### Método GET:

- Conteúdo é enviado para o servidor no campo URL:

`www.somesite.com/animalsearch?key=monkeys&max=10`

## Formato de mensagem HTTP: resposta

linha de status  
(protocolo,  
código de status,  
frase de status)

linhas de  
cabeçalho

dados, p.ex.,  
arquivo html  
solicitado

HTTP/1.1 200 OK  
Connection close  
Date: Thu, 06 Aug 1998 12:00:15 GMT  
Server: Apache/1.3.0 (Unix)  
Last-Modified: Mon, 22 Jun 1998 ...  
Content-Length: 6821  
Content-Type: text/html

dados dados dados dados ...

## 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

## Experimente você com HTTP (do lado cliente)

1. Use cliente telnet para seu servidor WWW favorito:

```
telnet www.ic.uff.br 80
```

Abre conexão TCP para a porta 80  
(porta padrão do servidor http) a www.ic.uff.br.  
Qualquer coisa digitada é enviada para a  
porta 80 do www.ic.uff.br

2. Digite um pedido GET HTTP:

```
GET /~celio/index.html HTTP/1.0
```

Digitando isto (deve teclar  
ENTER duas vezes), está enviando  
este pedido GET mínimo (porém  
completo) ao servidor http

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

## Cookies: manutenção do "estado" da conexão

Muitos dos principais sítios Web usam *cookies*

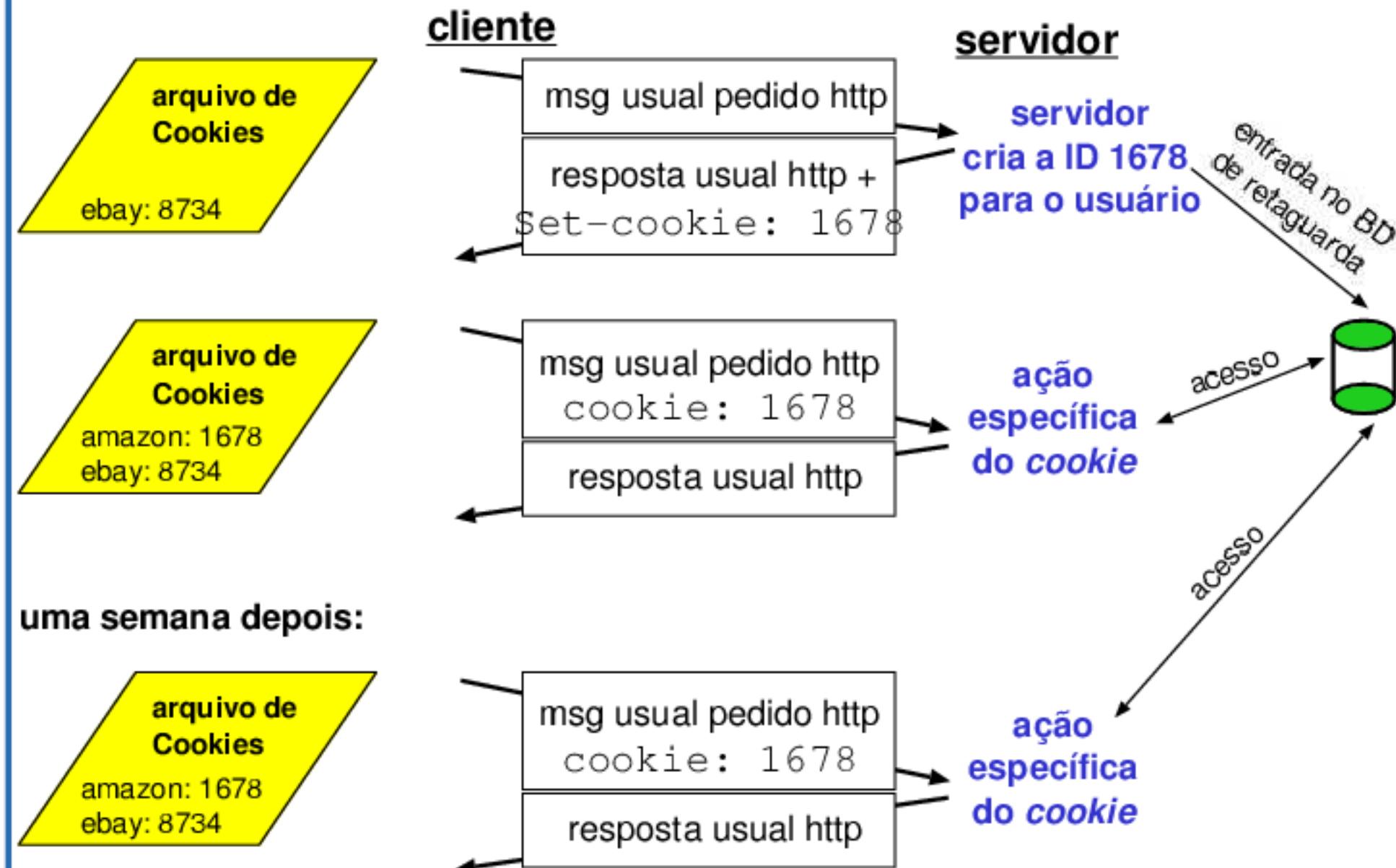
### Quatro componentes:

- 1) linha de cabeçalho do *cookie* na mensagem de resposta HTTP
- 2) linha de cabeçalho do *cookie* na mensagem de pedido HTTP
- 3) arquivo do *cookie* mantido no host do usuário e gerenciado pelo browser do usuário
- 4) BD de reaguarda no sítio Web

### Exemplo:

- Suzana acessa a Internet sempre do mesmo PC
- Ela visita um sítio específico de comércio eletrônico pela primeira vez
- Quando os pedidos iniciais HTTP chegam no sítio, o sítio cria uma ID única e cria uma entrada para a ID no BD de reaguarda

## Cookies: manutenção do "estado" (cont.)



## Cookies (continuação)

### O que os *cookies* podem obter:

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

Nota

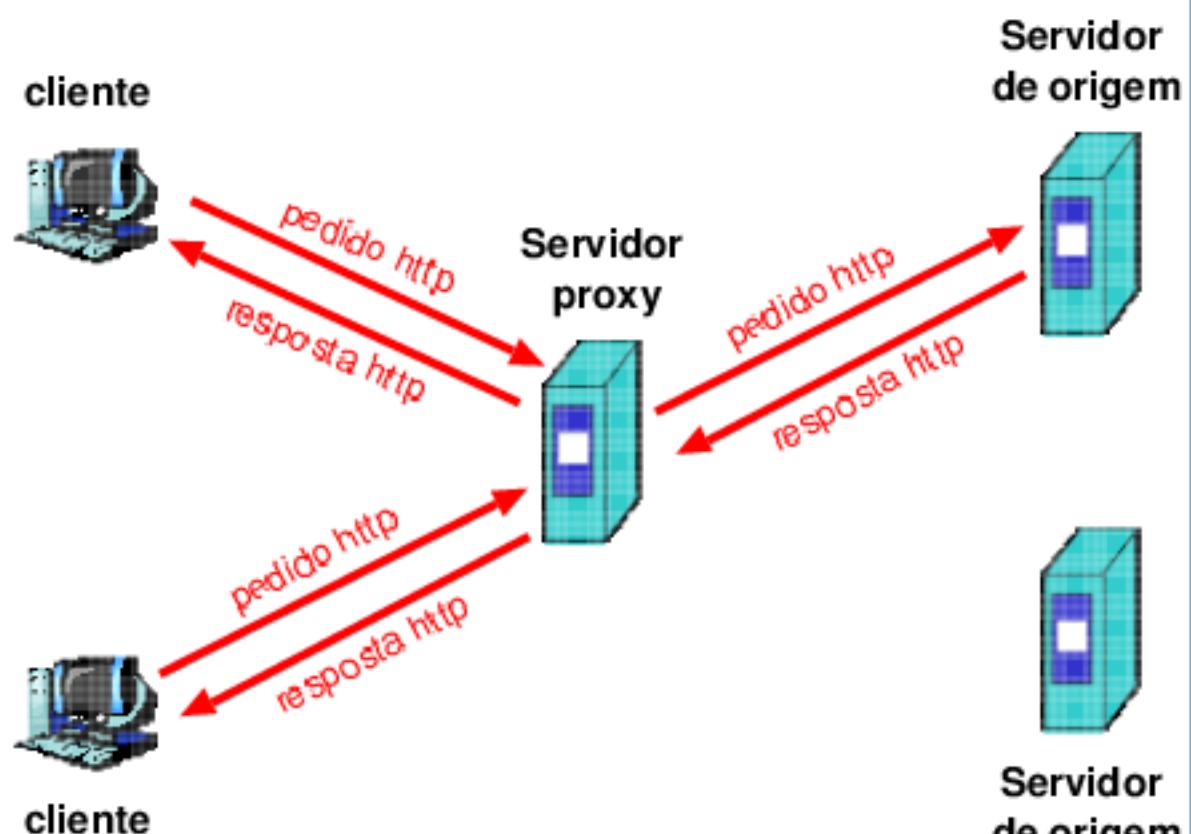
### Cookies e privacidade:

- cookies permitem que os sítios aprendam muito sobre você
- você pode fornecer nome e e-mail para os sítios
- mecanismos de busca usam redirecionamento e *cookies* para aprender ainda mais
- agências de propaganda obtêm perfil a partir dos sítios visitados

## Cache Web (servidor proxy)

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

- usuário configura browser: acessos Web via proxy
- cliente envia todos pedidos HTTP ao proxy
  - se objeto 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)

### Para que fazer cache Web?

- ❑ Redução do tempo de resposta para os pedidos do cliente
- ❑ Redução do tráfego no canal de acesso de uma instituição
- ❑ A Internet cheia de caches permitem que provedores de conteúdo "pobres" efetivamente forneçam conteúdo!

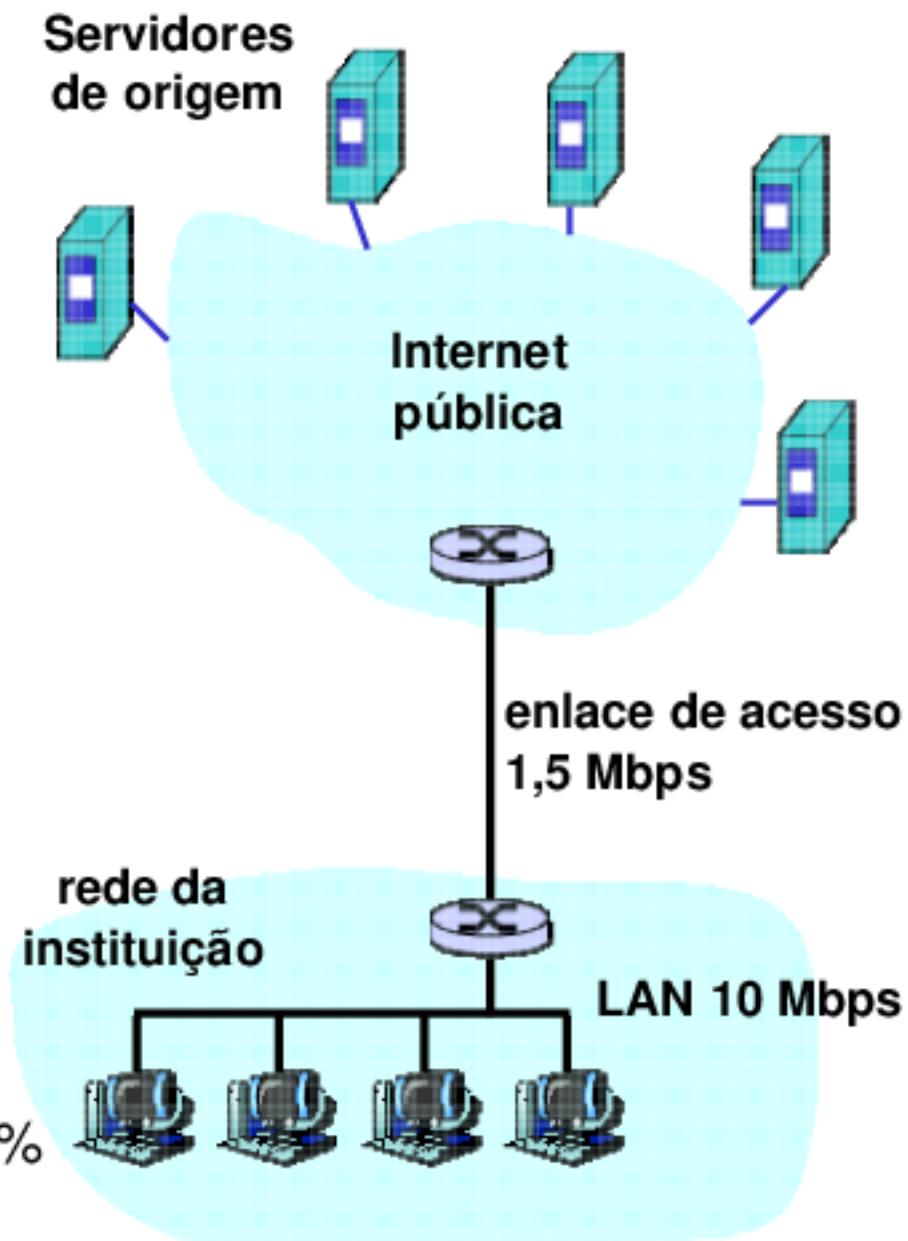
## Exemplo de cache (1)

### 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

### Conseqüências

- Utilização da LAN = 15%
- Utilização do canal de acesso = 100%
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + minutos + milisegundos



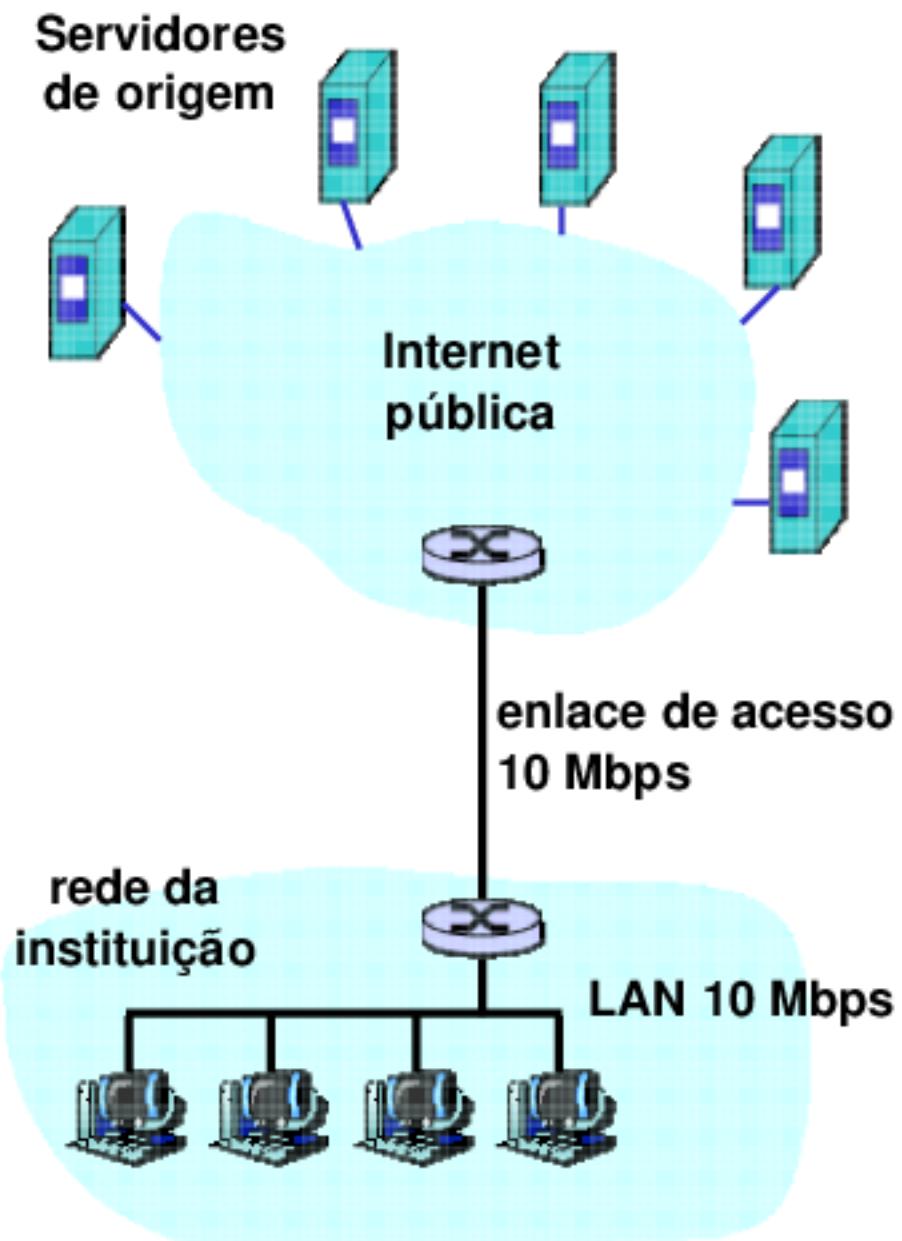
## Exemplo de cache (2)

### Solução em potencial

- Aumento da largura de banda do canal de acesso para, por exemplo, 10 Mbps

### Conseqüências

- Utilização da LAN = 15%
- Utilização do canal de acesso = 15%
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + mseg + mseg
- Freqüentemente este é uma ampliação cara



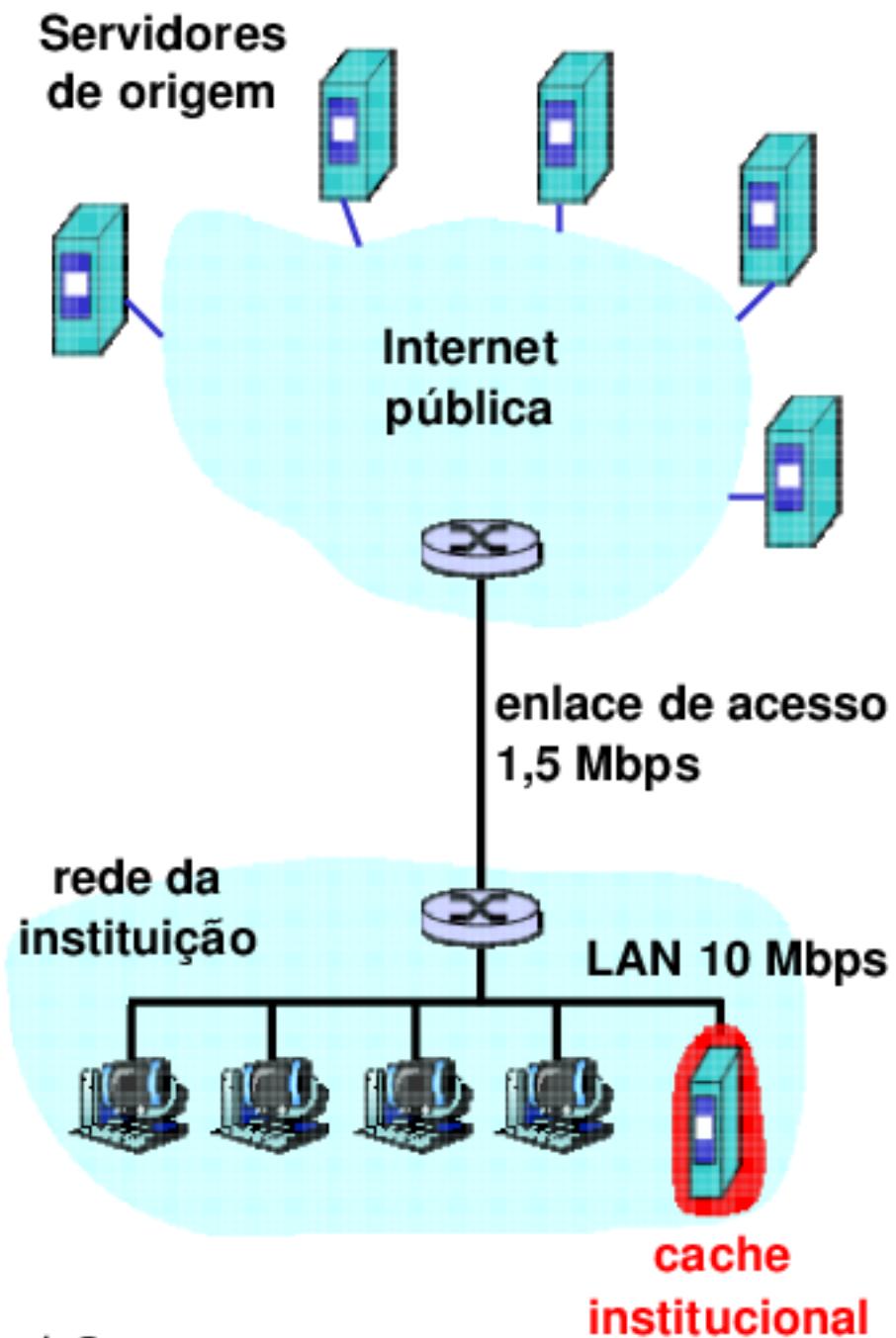
## Exemplo de cache (3)

### Instale uma cache

- Assuma que a taxa de acerto seja de 0,4

### Conseqüências

- 40% dos pedidos serão atendidos quase que imediatamente
- 60% dos pedidos serão servidos pelos servidores de origem
- Utilização do canal de acesso é reduzido para 60%, resultando em atrasos desprezíveis (ex. 10 mseg)
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN =  $0,6 \cdot 2 \text{ seg} + 0,6 \cdot 0,01 \text{ segs} + \text{msegs} < 1,3 \text{ segs}$



## GET condicional

- **Meta:** não enviar objeto se cliente já tem (no cache) versão atual

- cache: especifica data da cópia no cache no pedido http

If-modified-since:  
<date>

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

HTTP/1.0 304 Not Modified

