Redes de computadores

HyperText Transfer Protocol

Gabriel V C Candido
gabriel.candido@ifpr.edu.br

Instituto Federal do Paraná - Pinhais

INSTITUTO FEDERAL
Paraná
Campus Pinhais

Sumário

Cookies

Cache (servidor) e *Proxy*

Outras versões de HTTP

Relembrando...

- HTTP é cliente-servidor;
- HTTP usa TCP;
- O servidor HTTP usa a porta 80 por padrão;
- O cliente HTTP (navegador) conecta na porta 80 por padrão;
- Cliente e servidor trocam requisições e respostas.

Relembrando...

Requisição

```
GET /test.html HTTP/1.1
Host: 192.168.1.3
User-Agent: Mozilla/5.0
(X11; Linux x86_64; rv:128.0) Gecko/20100101
Accept-Encoding: gzip, deflate
```

Relembrando...

Resposta

```
HTTP/1.1 200 OK
Server: nginx
Date: Sun, 06 Apr 2025 23:28:05 GMT
Content-Type: text/html
Content-Length: 74
Last-Modified: Sun, 06 Apr 2025 21:41:18 GMT

<h1>Hey, this is an HTML page!</h1>\n
<a href="ref.html">This is a link</a>\n
```

Sumário

Cookies

Cache (servidor) e *Proxy*

Outras versões de HTTP

Motivação

- HTTP é *stateless*: não guarda informação sobre o cliente;
- Isso simplifica o código dos servidores e permite desempenho maior;
- Mas como os sites podem identificar usuários?

Cookies

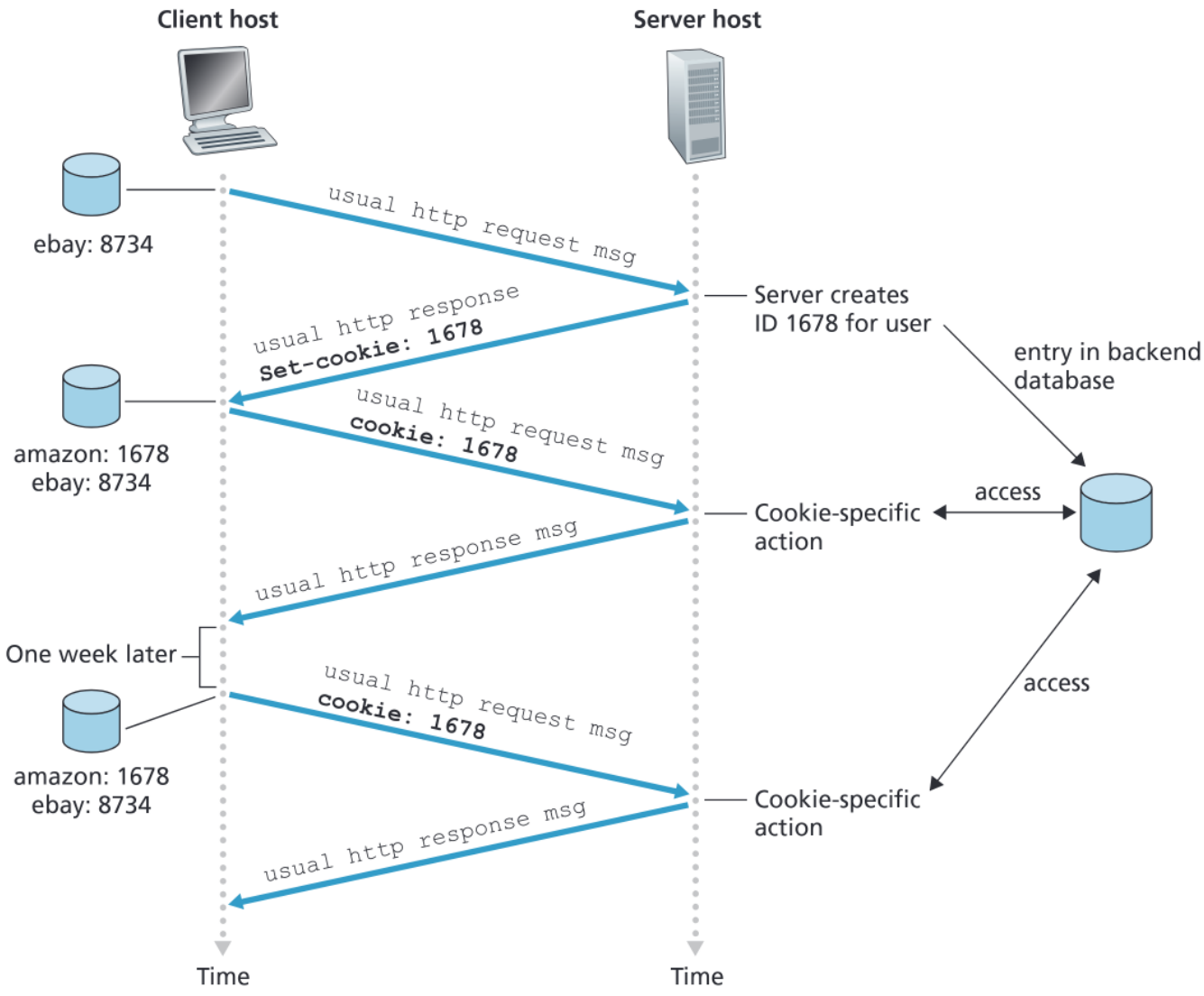

Figura: Cookies. Fonte: KR 2

Cookies

- Pode ser usado para *login*:
  1. Usuário envia as credenciais;
  2. Servidor gera um *cookie* com um *session ID*;
  3. Cliente usa o *cookie* em cada requisição;
  4. Ao deslogar, o servidor destroi a sessão e o *cookie*.

Cookies

- Cuidado com segurança:
  - Sempre usar a *flag Secure*: indica que o *cookie* só pode ser trocado em conexões criptografadas;
  - Sempre assinar (outra técnica de criptografia) o *cookie*;
  - Ainda pode ser capturado no cliente, pois está descriptografado.

Cookies

Existem outras formas de autenticação usando *tokens* (JWT, OAuth) que só ficam armazenados no cliente.

Ambas as abordagens possuem vantagens e desvantagens e são adequadas para cenários diferentes.

Sumário

Cookies

Cache (servidor) e *Proxy*

Outras versões de HTTP

Servidor *proxy/cache*

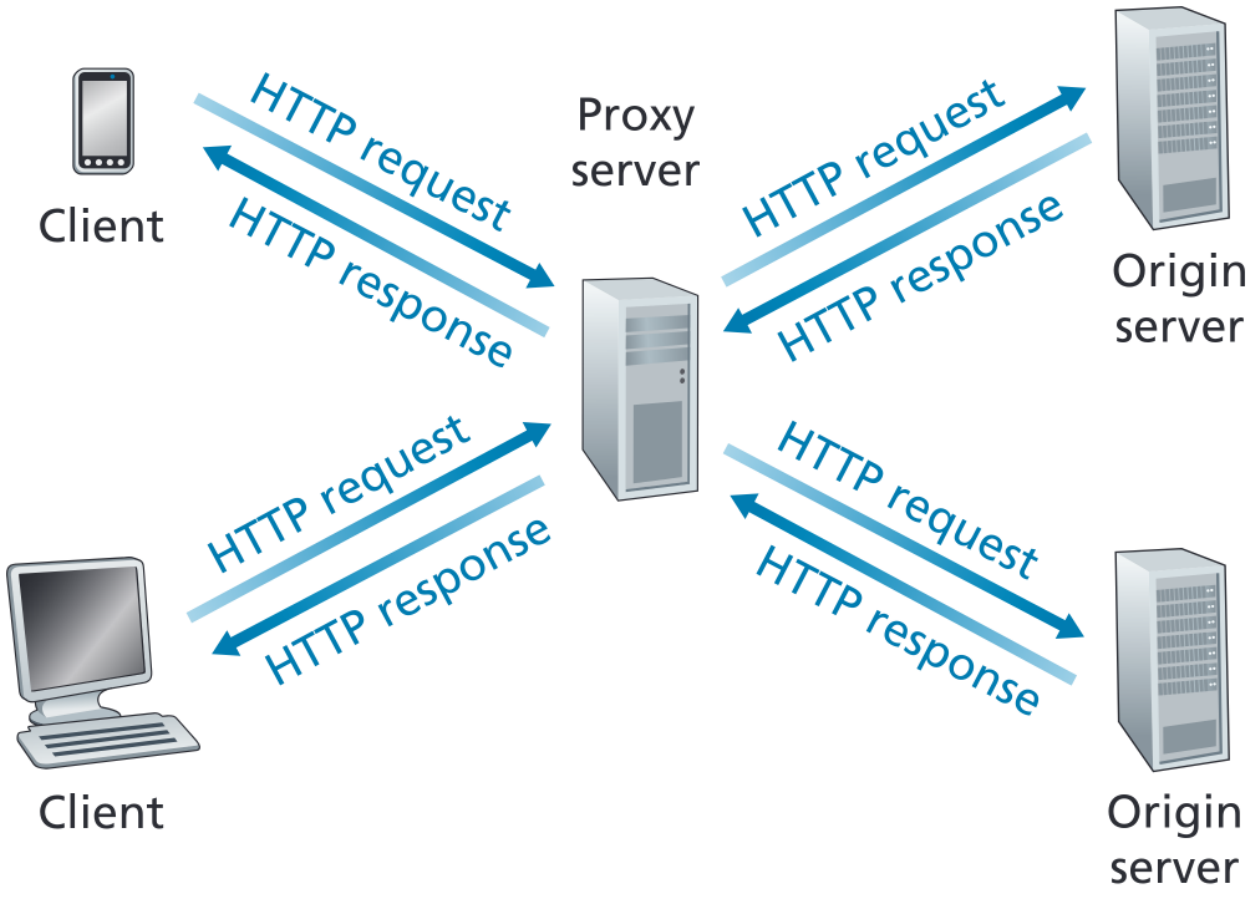

Figura: Clientes fazendo requisições através de um servidor *proxy*. Fonte: KR 2

Como funciona?

1. O navegador é configurado para acessar a Internet através do *proxy*;
2. O navegador estabelece uma conexão TCP com o *proxy* e faz a requisição;
3. Se o *proxy* possui uma cópia local do objeto requisitado, o *proxy* retorna o objeto como resposta ao cliente;
4. Se o *proxy* não possui o objeto, o *proxy* faz uma **nova conexão** com o servidor que possui o objeto e faz a requisição;
5. Quando o *proxy* recebe o objeto do servidor, ele armazena uma cópia local (*cache*) e envia o objeto como resposta da requisição original do cliente.

Motivação

- É interessante notar que o *proxy* atua como cliente e servidor ao mesmo tempo.

Para quê fazer isso?

- Reduzir a latência de resposta, principalmente em conexões mais lentas;
- Reduzir a quantidade de tráfego de uma rede com a Internet.

Exemplo

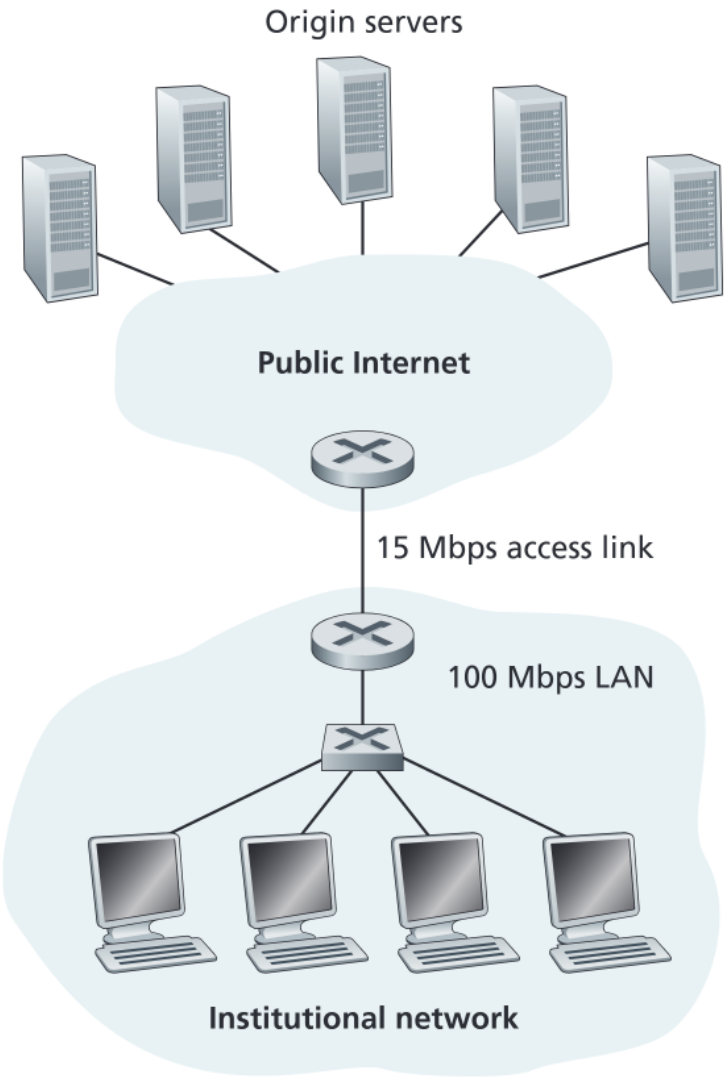

Figura: Exemplo de rede com gargalo (*bottleneck*). Fonte: KR 2

Exemplo

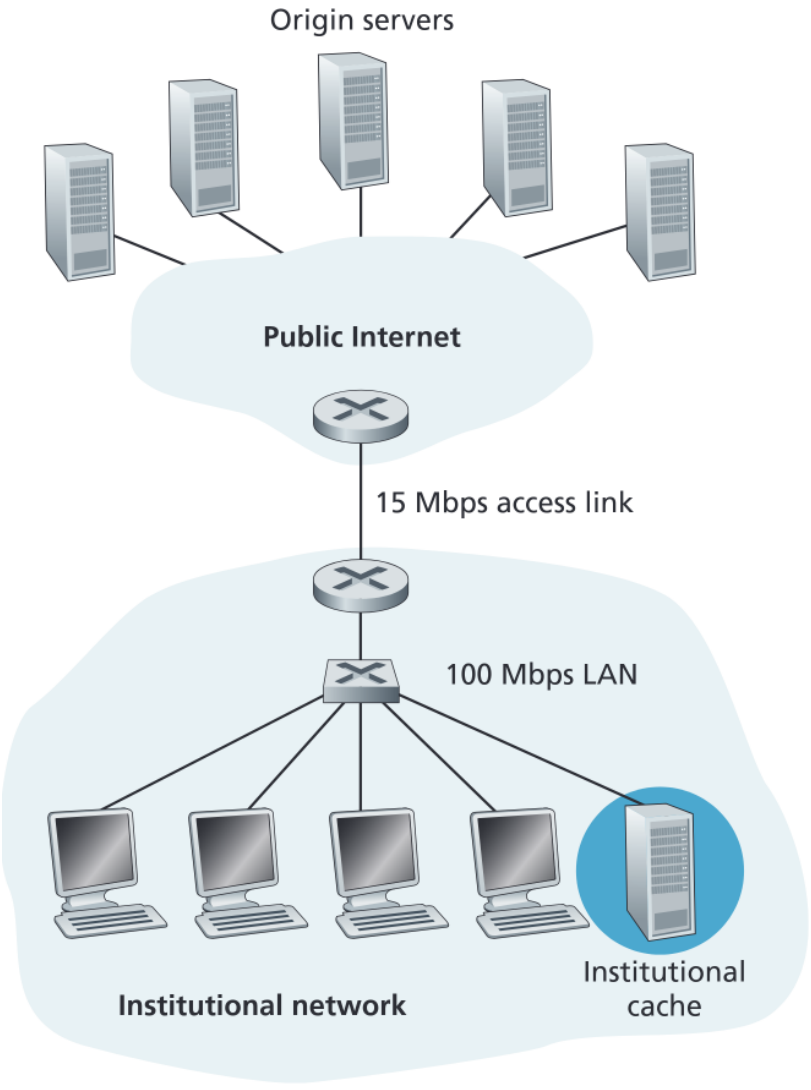

Figura: Rede local usando *cache*. Fonte: KR 2.

Mais motivação

Content Distribution Networks (CDN):

Google, Netflix, Akamai, ...

Cache local

O nosso navegador também faz *cache* dos objetos!

```
$ du -hd1 ~/.cache/mozilla/firefox/
1.1G /home/gabriel/.cache/mozilla/firefox/
     hcr0av9f.default-release
```

Problema

E se o objeto que está na *cache* for atualizado?

Conditional GET: a requisição inclui um cabeçalho de data para verificar se o objeto em *cache* está ativo ou velho (e deve ser atualizado).

Conditional GET

1. O cliente faz a requisição normalmente;

2. Ao receber o objeto, a *cache* armazena uma cópia do objeto e sua data de modificação:

```
Last-Modified: Wed, 9 Sep 2015 09:23:24
```

3. Mais tarde, o cliente requisita o mesmo objeto à *cache*;

Conditional GET

4. A *cache* faz a requisição ao servidor original, pois o conteúdo pode ter sido alterado:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

Conditional GET

5.a. Se o objeto foi alterado, o servidor original reenvia o objeto e a *cache* atualiza sua cópia;

5.b. Se o objeto não foi alterado, o servidor não reenvia o objeto e a cópia da *cache* é usada:

```
HTTP/1.1 304 Not Modified
Date: Sat, 10 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```

Sumário

Cookies

Cache (servidor) e *Proxy*

Outras versões de HTTP

HTTP/2

1. Priorização de mensagens;
2. Uso melhor das conexões (principalmente para grandes objetos);
3. Compatibilidade com HTTP/1.1.

- Navegadores já suportam;
- $\sim$ metade das páginas ainda não utilizam HTTP/2.

HTTP/3

1. Várias mudanças e características novas;
2. Usa QUIC e UDP como transporte (e não TCP): *muita* coisa muda;
3. Também implementa melhorias do HTTP/2.

HTTPS

1. Utiliza TLS para criptografar conexões;
2. Certificados podem garantir que uma página é verdadeira;
3. Dados criptografados não podem ser lidos mesmo quando são interceptados;
4. Não podemos ler o conteúdo dos pacotes no Wireshark;
5. Usa a porta 443 por padrão.